INFORMAČNÍ BULLETIN

České statistické společnosti

Ročník 36, mimořádné číslo, prosinec 2025

Obsah

Informační bulletin České statistické společnosti vychází čtyřikrát do roka v českém vydání. Příležitostně i mimořádné české a anglické číslo. Vydavatelem je Česká statistická společnost, IČ 00550795, adresa společnosti je Na padesátém 81, 100 82 Praha 10. Evidenční číslo registrace vedené Ministerstvem kultury ČR dle zákona č. 46/2000 Sb. je E 21214. Časopis je sázen v programu TEX, ve formátu LuaHBTEX s písmy balíku CSfonts.

The Information Bulletin of the Czech Statistical Society is published quarterly.
The contributions in the journal are published in English, Czech and Slovak languages.

Předsedkyně společnosti: Ing. Martina Litschmannová, Ph.D., Katedra aplikované matematiky, Fakulta elektrotechniky a informatiky, Vysoká škola báňská – Technická univerzita Ostrava, 17. listopadu 2172/15, 708 33 Ostrava-Poruba, `Martina.Litschmannova@vsb.cz`.

Redakce: prof. RNDr. Gejza Dohnal, CSc. (šéfredaktor), prof. RNDr. Jaromír Antoch, CSc., doc. RNDr. Zdeněk Karpíšek, CSc., RNDr. Marek Malý, CSc., doc. RNDr. Jiří Michálek, CSc., prof. Ing. Jiří Militký, CSc., doc. Ing. Iveta Stankovičová, PhD., doc. Mgr. Ondřej Vencálek, Ph.D.

Redaktor časopisu: doc. Mgr. Ondřej Vencálek, Ph.D., `ondrej.vencalek@upol.cz`.
Informace pro autory jsou na stránkách společnosti, `http://www.statspol.cz/`.

DOI: 10.5300/IB, http://dx.doi.org/10.5300/IB
ISSN 1210–8022 (Print), ISSN 1804–8617 (Online)

PŘEDMLUVA
FOREWORD

Redakce

Vážené kolegyně, vážení kolegové,

od roku 2010 pomáhám s konferencí OSSConf v Žilině. Akce se koná na akademické půdě s možností využít počítačových učeben během akce. Touto cestou vznikají workshopy, některé z nich v tomto čísle naleznete. Tři z nich proběhly v roce 2025, obvykle trvaly 2 až 3 hodiny.

Co však považuji za důležité je, že i kdyby nebyla počítačová učebna na akci k dispozici, dá se k výuce příchozích použít jejich notebooky. Webové stránky workshopů dávají možnost se k tématům vrátit, dodělat si je či si je znovu zkusit doma. Přikládám medailónky slovenských tvůrců.

Autorka: Zuzana Tkáčová

Som (zatiaľ ešte stále) nadšená učiteľka informatiky a programovania na SŠ. Vyštudovala som AI a v súčasnosti pôsobím v tejto oblasti ako lektorka, metodička, školiteľka učiteľov, publikujem a vediem tím pre AI vo vzdelávaní v Národnom centre pre digitálnu transformáciu vo vzdelávaní. Moje superschopnosti: creative computing & physical computing.

Autor: Mirek Biňas

Odborný asistent na Katedre počítačov a informatiky, Technickej univerzite v Košiciach (TUKE), kde učí „detiská“ programovať najmä v jazyku C a Python.

Stále dúfa, že keď vyrastie, bude z neho Linuxák. Alebo Batman. Webová stránka autora: bletvaska.github.io.

Poněvadž jsou autoři skromní, zmíním pár postřehů.

Na premiéře soutěže *Učiteľ Slovenska* získala Zuzana Tkáčová první místo, a s úsměvem můžeme říci, že získala titul mediální hvězdy za AI a nanotechnologie. Mirek Biňas je její kolega v práci, aktivní programátor a vývojář s neskutečným zápalem pro věc. I kdyby ta věc byla rozbitá, tak se ji bude snažit opravit.

Do tištěného bulletinku se nám nevešly dvě zprávy o OSSConf, a poněvadž stárnou, přikládáme je zde.

Ve Slavkově u Brna, na Mezinárodní den studentstva 17. 11. 2025,
Pavel Stříž

KREATÍVNE PROGRAMOVANIE S P5.JS (... A CHATGPT)

CREATIVE CODING WITH P5.JS (... AND CHATGPT)

Zuzana Tkáčová

E-mail: zuzkat152@gmail.com

Kreatívne programovanie (Creative Coding) je forma programovania, ktorej cieľom nie je len riešiť praktické úlohy, ale tvorba vizuálnych, zvukových a interaktívnych projektov. Ide o programovanie ako prostriedok kreatívneho vyjadrenia. Vhodné je pre tvorbu animácií, vizualizácií, interaktívnych aplikácií alebo vizuálne atraktívnych efektov. Hlavným princípom je experimentovanie a kreativita, pričom pravidlá sú flexibilné a prispôsobiteľné konkrétnemu projektu. V klasickom programovaní sa často sústredíme na funkčnosť a efektívnosť riešenia, zatiaľ čo v creative coding ide o **estetický a interaktívny aspekt výsledku**.

p5.js je javascriptová open source knižnica špeciálne navrhnutá pre kreatívne programovanie. Umožňuje jednoducho vytvárať vizuálne a interaktívne projekty priamo vo webovom prehliadači. Poskytuje jednoduché funkcie na kreslenie tvarov, prácu s farbami, animácie a interaktivitu s používateľom. Funguje ako nástroj https://donorbox.org/back-to-school-805292 pre digitálne umenie – namiesto tradičných umeleckých pomôcok používa kód. Projekty vytvorené v p5.js môžu byť interaktívne, napríklad reagovať na myš, klávesnicu alebo čas.

Výhody pre začiatočníkov:

1. Rýchle vizuálne výsledky – zmeny kódu sú okamžite viditeľné.
2. Učenie programovania cez tvorbu – študenti vidia priamy účinok svojho kódu.
3. Kreatívna sloboda – možné je vytvárať od abstraktných vizualizácií po interaktívne animácie.

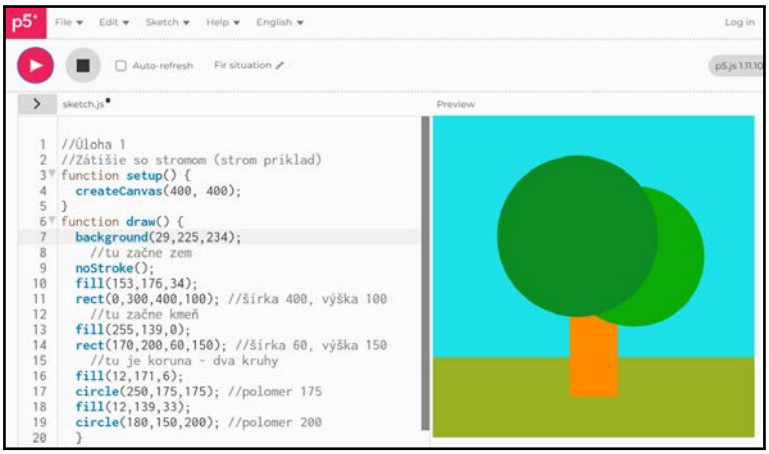
Knižnica je dostupná online (spolu s referenčnou príručkou, tutoriálmi, príkladmi a samotným web editorom kódu) na stránke https://p5js.org/. Webový editor funguje aj bez prihlásenia (bez možnosti ukladania kódu). V jeho ľavej časti sa píše samotný kód a v pravej časti sa zobrazí po spustení (kliknutím na tlačidlo šípky PLAY) grafický náhľad bežiaceho kódu:

Základná štruktúra kódu:

- `function setup()` sa vykoná len raz (pri spustení kódu),
- `function draw()` sa vykonáva opakovane v implicitnom cykle, čo je užitočné pri vytváraní animovaných efektov.

Ako na to?

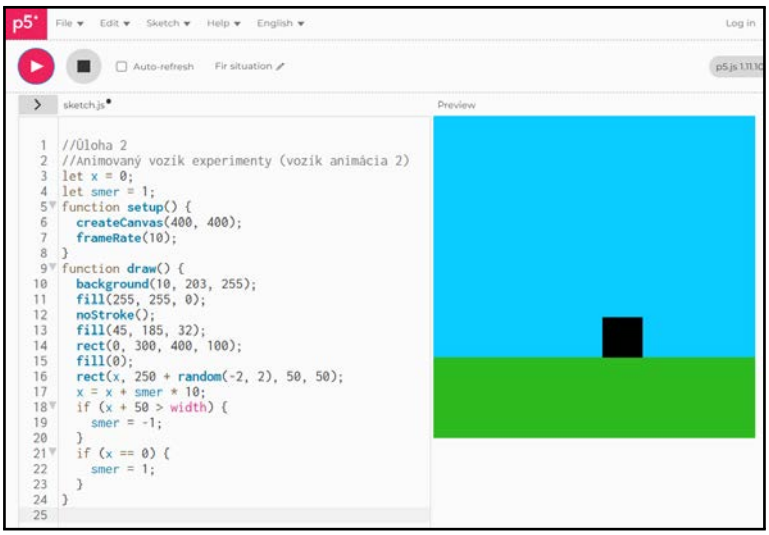
Začína sa statickým obrázkom, vytvoreným len zo základných geometrických tvarov – obdĺžnikmi a kruhmi (`rect` a `circle`) a ich farebnými vlastnosťami – výplňou a vypnutím obvodovej kontúry (`fill` a `noStroke`). Výplň môže mať jednu číselnú hodnotu od 0 do 255 (odtiene šedej) alebo tri hodnoty od 0 do 255 (intenzity farieb RGB); niekedy má ešte navyše nepovinný číselný parameter alfa (priehľadnosť). Počiatok súradnicového systému je `v ľavom hornom rohu` kresliaceho plátna.



```
//Úloha 1
//Zátišie so stromom (strom príklad)
function setup() {
 createCanvas(400, 400);
}
function draw() {
 background(29,225,234);
 //tu začne zem
 noStroke();
 fill(153,176,34);
 rect(0,300,400,100); //šírka 400, výška 100
 //tu začne kmeň
 fill(255,139,0);
 rect(170,200,60,150); //šírka 60, výška 150
 //tu je koruna - dva kruhy
 fill(12,171,6);
 circle(250,175,175); //polomer 175
 fill(12,139,33);
 circle(180,150,200); //polomer 200
 }
```

Pokračuje sa jednoduchou animáciou predstavujúcou kmitaný pohyb krabice vpravo a vľavo. Na vytvorenie samotnej animácie je potrebné si deklarovať premennú `x` (aktuálna vzdialenosť balíčka od ľavého okraja) a pomocnú premennú smer, ktorá nadobúda len dve hodnoty (1 pre pohyb balíčka smerom vpravo, -1 pre pohyb balíčka smerom vľavo). Rýchlosť animácie sa nastavuje pomocou hodnoty `frameRate` (počet opakovaní za sekundu). Na opakovaný smer pohybu balíčka je potrebné pri dosiahnutí okraja kresliaceho plátna pomocou `if` meniť hodnotu smeru medzi 1 a -1.

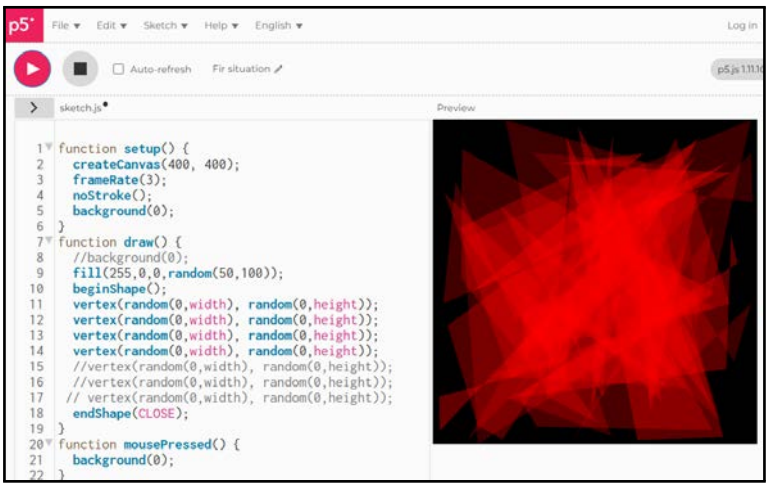
```
//Úloha 2
//Animovaný vozík experimenty
let x = 0;
let smer = 1;
function setup() {
 createCanvas(400, 400);
 frameRate(10);
}
function draw() {
 background(10, 203, 255);
 fill(255, 255, 0);
```




```
 noStroke();
 fill(45, 185, 32);
 rect(0, 300, 400, 100);
 fill(0);
 rect(x, 250 + random(-2, 2), 50, 50);
 x = x + smer * 10;
 if (x + 50 > width) {smer = -1;}
 if (x == 0) {smer = 1;}
}
```

Ďalší animovaný príklad je vytvorenie efektu organzy opakovaným vrstvením polotransparentných nepravidelných štvoruholníkov vytváraných pomocou `beginShape` a `endShape` požadovaným počtom jednotlivých vrcholov (`vertex`). Náhodné čísla `random` sú generované v rozpätí požadovaného intervalu – pre alfa kanál postačuje rozpätie 50 až 100, pre súradnice vrcholov je rozpätie od 0 po šírku plátna (`width`) alebo od 0 po výšku plátna (`height`). Do kódu je možné vytvoriť aj funkciu `mousePressed`, ktorá bude zavolaná v okamihu kliknutia myšou na plátne a vymaže nakreslené objekty.

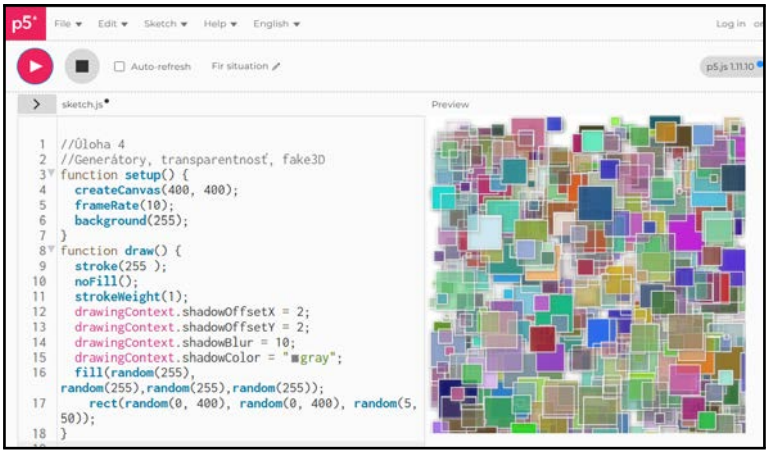
```
//Úloha 3
//Organza
function setup() {
 createCanvas(400, 400);
```




```
 frameRate(3);
 noStroke();
 background(0);
}
function draw() {
 //background(0);
 fill(255,0,0,random(50,100));
 beginShape();
 vertex(random(0,width), random(0,height));
 vertex(random(0,width), random(0,height));
 vertex(random(0,width), random(0,height));
 vertex(random(0,width), random(0,height));
 //vertex(random(0,width), random(0,height));
 //vertex(random(0,width), random(0,height));
 //vertex(random(0,width), random(0,height));
 endShape(CLOSE);
}
function mousePressed() {
 background(0);
}
```

Kreatívnou výzvou je vytvorenie grafického generátora abstraktných geometrických obrazcov. Ich farbu a výplň je možné upraviť (alebo vypnúť)

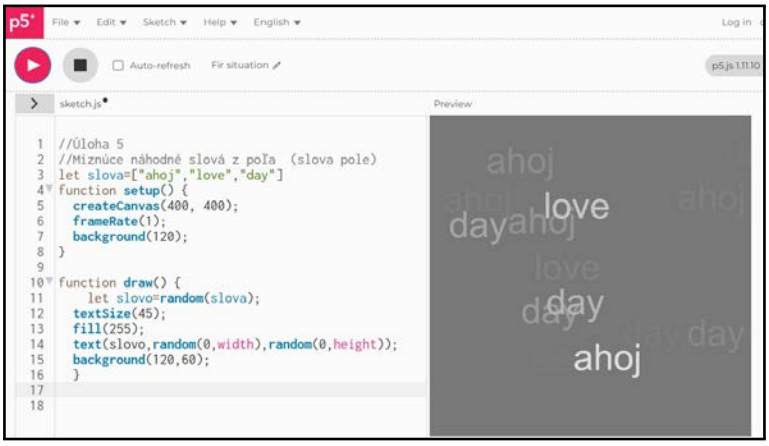
pomocou `noFill`, `stroke` a `strokeWeight`. Okrem týchto parametrov je ešte možné vylepšiť generátor pridaním tieňa na plátno (`drawingContext`).




```
//Úloha 4
//Generátory, transparentnosť, fake3D
function setup() {
 createCanvas(400, 400);
 frameRate(10);
 background(255);
}
function draw() {
 stroke(255 );
 noFill();
 strokeWeight(1);
 drawingContext.shadowOffsetX = 2;
 drawingContext.shadowOffsetY = 2;
 drawingContext.shadowBlur = 10;
 drawingContext.shadowColor = "gray";
 fill(random(255), random(255),random(255),random(255));
 rect(random(0,400), random(0,400), random(5,50));
}
```

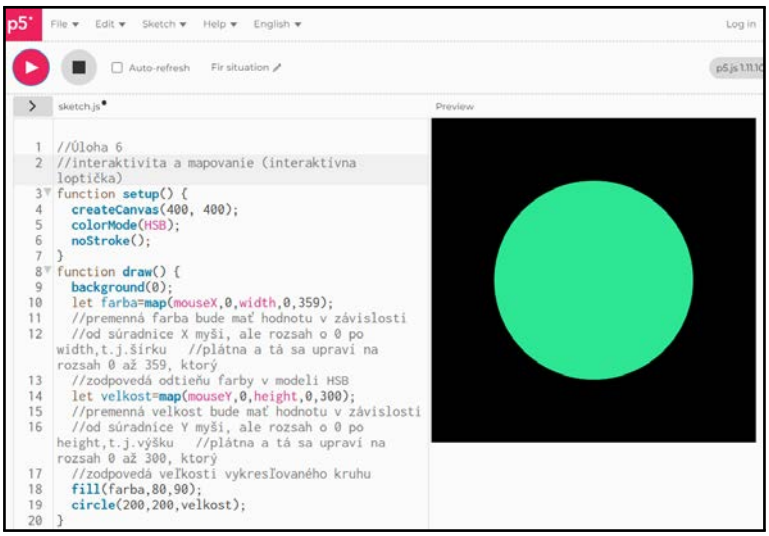
Kreatívne pokusy sa dajú robiť aj pomocou textu vytvorením jednoduchej animácie náhodne umiestnených slov na plátne a ich následným premaľovaním vysokotransparentným pozadím. Slová sa vyberajú náhodne zo zoznamu slov (`slova`), ktorý je potrebné mať vytvorený hneď na začiatku kódu. Na

náhodný výber slov zo zoznamu sa používa taktiež `random`, tentokrát ale jeho parametrom je meno zoznamu, z ktorého sa majú náhodne položky vyberať.



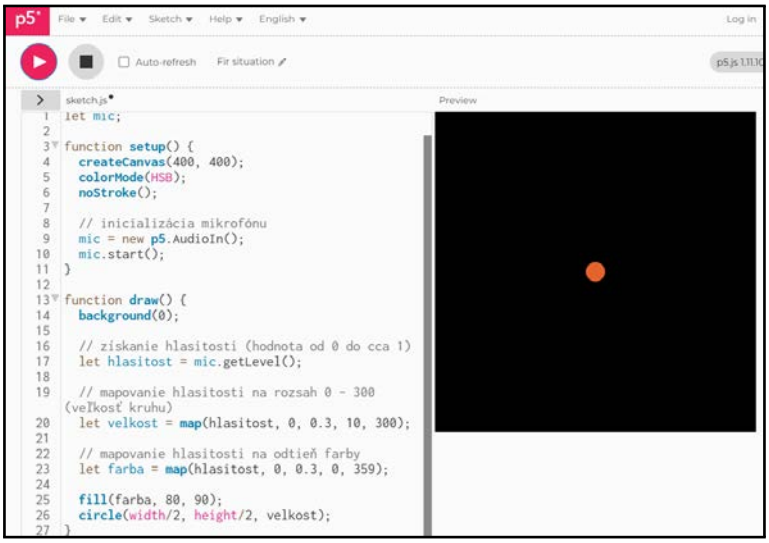

```
//Úloha 5
//Miznúce náhodné slová z poľa (slova pole)
let slova=["ahoj","love","day"]
function setup() {
 createCanvas(400, 400);
 frameRate(1);
 background(120);
}
function draw() {
 let slovo=random(slova);
 textSize(45);
 fill(255);
 text(slovo,random(0,width),random(0,height));
 background(120,60);
 }
```

Pri programovaní je možné využiť aj ChatGPT. Začať sa dá s pripraveným kódom, v ktorom je nachystaný aj farebný model HSB (bližšie info https://www.learnui.design/blog/the-hsb-color-system-practicioners-primer.html). Mapovaním (`map`) sa dosiahne zmena hodnoty premennej z jedného rozsahu do iného. V tomto príklade sa farba pozadia mení podľa aktuálnej X súradnice myši a veľkosť loptičky podľa Y súradnice.




```
//Úloha 6
//interaktivita a mapovanie (interaktívna loptička)
function setup() {
 createCanvas(400, 400);
 colorMode(HSB);  noStroke();
}
function draw() {
 background(0);
 let farba=map(mouseX,0,width,0,359);
 //premenná farba bude mať hodnotu v závislosti
 //od súradnice X myši, ale rozsah o 0 po width,t.j.šírku
 //plátna a tá sa upraví na rozsah 0 až 359, ktorý
 //zodpovedá odtieňu farby v modeli HSB
 let velkost=map(mouseY,0,height,0,300);
 //premenná velkost bude mať hodnotu v závislosti
 //od súradnice Y myši, ale rozsah o 0 po height,t.j.výšku
 //plátna a tá sa upraví na rozsah 0 až 300, ktorý
 //zodpovedá veľkosti vykresľovaného kruhu
 fill(farba,80,90);
 circle(200,200,velkost);
}
```

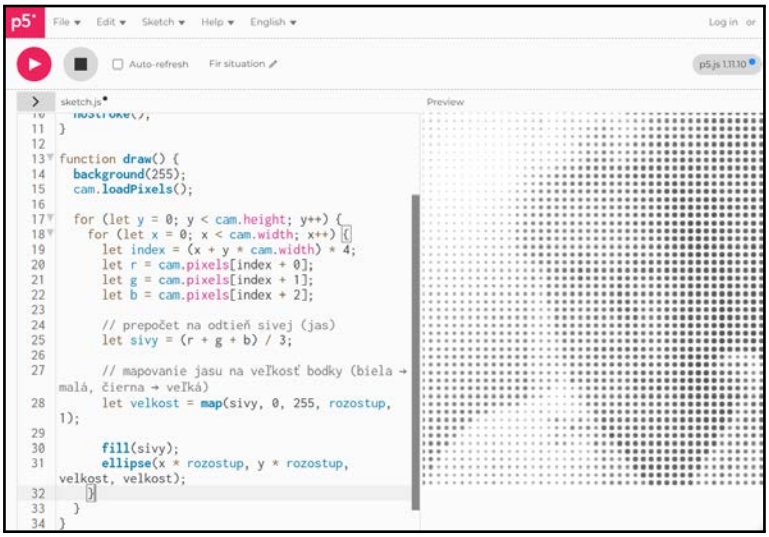
Po vytvorení a otestovaní prvého funkčného kódu ho vložíme do ChatGPT. Pomocou promptu (*Uprav tento kód tak, aby reagoval na vstup z mikrofónu*) môžeme získať návrh upraveného kódu na živú interakciu prostredníctvom mikrofónu, pri ktorej sa veľkosť vykresľovanej loptičky mení pomocou intenzity okolitého zvuku – je potrebné deklarovať premennú `mic`, inicializovať mikrofón (`new p5.AudioIn`) a spustiť ho (`mic.start`). Do premennej `hlasitost` sa potom bude plynule z mikrofónu načítavať intenzita zvuku (`mic.getLevel`) a premapuje sa na vhodný rozsah veľkosti loptičky.




```
let mic;
function setup() {
 createCanvas(400, 400);
 colorMode(HSB);
 noStroke();
 // inicializácia mikrofónu
 mic = new p5.AudioIn();
 mic.start();
}
function draw() {
 background(0);
 // získanie hlasitosti (hodnota od 0 do cca 1)
 let hlasitost = mic.getLevel();
 // mapovanie hlasitosti na rozsah 0-300 (veľkosť kruhu)
```

```
  let velkost = map(hlasitost, 0, 0.3, 10, 300);
  // mapovanie hlasitosti na odtieň farby
  let farba = map(hlasitost, 0, 0.3, 0, 359);
  fill(farba, 80, 90);
  circle(width/2, height/2, velkost);
}
```

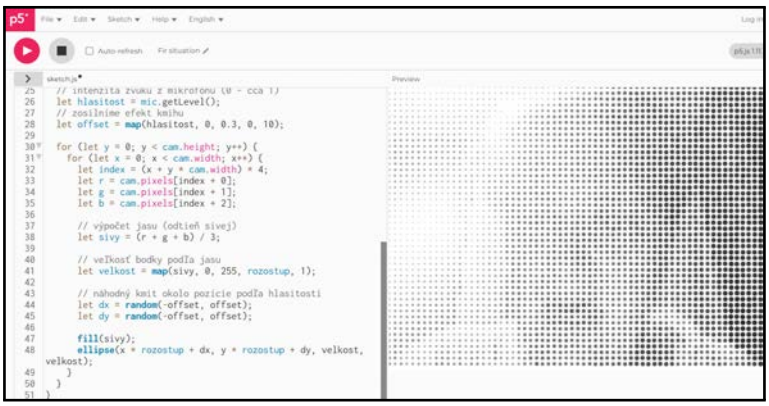
Kód môžeme začať generovať aj od začiatku pomocou ChatGPT zadaním základného promptu (*Vytvor kód v p5.js pre prácu s webkamerou, kde obraz v pixeloch bude zmenený na bodky. Veľkosť bodky a jej farba bude v odtieňoch sivej a závisí od intenzity sivej farby – biele budú najmenšie, čierne najväčšie*). Vytvorený kód je potrebné vždy dôkladne otestovať.



A môžeme vymýšľať jeho modifikácie – napríklad pomocou promptu (*Pridaj do kódu aj využitie mikrofónu – podľa intenzity hlasitosti budú bodky kmitať okolo svojich pozícií*).

V programe je možné využívať aj knižnicu `ml5.js` (dostupná online https://ml5js.org/). Je to javascriptová open source knižnica, ktorá sprístupňuje strojové učenie jednoduchým a kreatívnym spôsobom priamo v prehliadači. Je postavená na TensorFlow.js, ale jej rozhranie je omnoho prístupnejšie pre začiatočníkov a tvorcov interaktívneho obsahu. Pomocou ml5.js sa dajú ľahko využiť predtrénované modely napríklad na:

- rozpoznávanie obrázkov (napr. detekcia objektov cez webkameru),
- rozpoznávanie zvuku alebo reči,



- klasifikáciu textov,
- generovanie obrázkov alebo textu,
- prácu s pozíciou tela a gestami (napr. sledovanie pohybu človeka cez kameru).

Záver

Tieto aj ďalšie príklady pre prácu s p5.js je možné nájsť v postupne vytváranej interaktívnej učebnici *Techniky kreatívneho programovania* (voľne dostupnej na stránke `https://shorturl.at/y0BfV`) , ktorá slúži ako učebnica pre rovnomenný predmet vyučovaný na Škole umeleckého priemyslu v Košiciach.

ESP32 S JAZYKOM MICROPYTHON
WORKSHOP: ESP32 WITH MICROPYTHON

Mirek Biňas

E-mail: miroslav.binas@tuke.sk

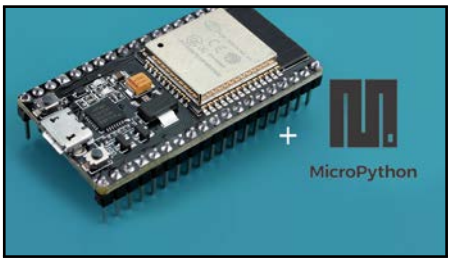
Mikrokontrolér *ESP32* je cenovo dostupné a namakané zariadenie vhodné pre oblasť *IoT* vybavené *WiFi* a *Bluetooth LE*. Čo je však úplne fantastické, má dostatok pamäte na to, aby ste do neho nahrali firmvér s jazykom *MicroPython*. Na tomto workshope si spolu vytvoríme jednoduché *IoT* riešenie, na ktorom ukážeme silu mikrokontroléra *ESP32* a jednoduchosť jeho programovania vďaka jazyku *MicroPython*.

Odporúčaný čas: 180 minút.

Upozornenie: Webinár nebude dostatočne dlhý na to, aby naše výsledné riešenie spĺňalo kritériá uvedené v knihe Čistý kód. Zameriame sa preto viac na WOW efekt celého výsledku. O detailoch/rozporoch/vylepšeniach sa môžeme porozprávať osobne mimo workshop-u. ;)

Ciele

1. Naučiť sa základy práce v REPL režime jazyka *MicroPython*.
2. Naučiť sa pracovať s vnútornými senzormi mikrokontroléra *ESP32*.
3. Naučiť sa pripojiť mikrokontrolér *ESP32* do internetu pomocou zabudovaného *WiFi* modulu.
4. Naučiť sa publikovať údaje z mikrokontroléra pomocou komunikačného protokolu *MQTT*.
5. Naučiť sa základy používania modulu `urequests` pre prácu s HTTP protokolom.



Obr. 1: ESP32 Running MicroPython

Krok 1. Čo budeme potrebovať?

Ešte predtým, ako sa pustíme do tvorby aplikácie, si pripravíme prostredie pre prácu. Konkrétne budeme potrebovať:

- editor kódu Thonny (alebo ľubovoľný iný editor kódu jazyka Python),
- dosku s mikrokontrolérom *ESP32*, zapojenie podľa schémy a USB kábel na prepojenie dosky s počítačom,
- magnet,
- LED diódu,
- voľný vodič typu M-F (na detekovanie dotyku),
- prekopírovať na mikrokontrolér súbory `playground.py`, `helpers.py` a vytvoriť prázdny súbor `workshop.py`, do ktorého budeme zapisovať kód výsledného riešenia,
- kľúč pre prístup k HTTP REST API služby `openweathermap.org` (pre workshop stačí jeden, takže stačí, ak sa zaregistruje len inštruktor a vyzdieľa svoj kľúč s ostatnými; registrácia je zdarma),
- nahratý alternatívny firmwér s podporou protokolu ESP-NOW z tejto adresy.

Ak používate OS Windows a chcete pracovať s mikrokontrolérom *ESP32*, musíte si nainštalovať ešte ovládač pre *CP210x USB to UART Bridge* napríklad odtiaľto.

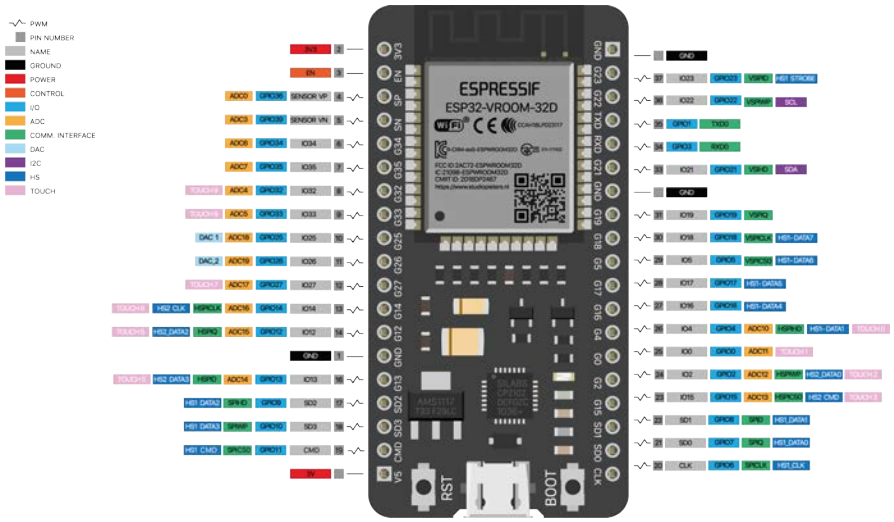
Krok 2. Predstavenie mikrokontroléra ESP32

Samotný mikrokontrolér obsahuje tieto senzory:

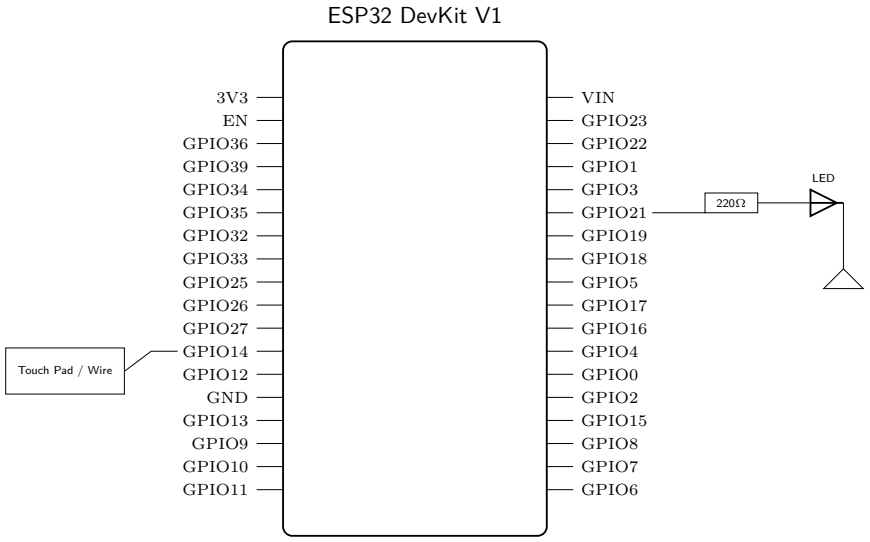
- hallova sonda/senzor,
- kapacitný dotykový senzor,
- senzor vnútornej (pracovnej) teploty.

A presne na prácu s týmito senzormi sa pozrieme počas tohto workshopu. Vytvoríme si jednoduchý scenár, pomocou ktorého budeme monitorovať teplotu v miestnosti a vstup do nej:

- Teplomer bude predstavovať aktuálnu teplotu v miestnosti. Tú budeme v (ne)pravidelných intervaloch posielať pomocou protokolu MQTT ďalšej službe.
- Hallovu sondu budeme používať ako snímač otvorených/zatvorených dverí. Na tento účel nám pomôže ešte externý magnet. Priložením magnetu k hallovej sonde budeme simulovať zatvorenie dverí. Oddialením



Obr. 2: Rozloženie pinov na doske s mikrokontrolérom ESP32



Obr. 3: Schéma konkrétneho zapojenia

magneta sa dvere otvoria. A okrem toho – keď budú dvere otvorené, rozsvieti sa LED dióda. Keď sa naopak dvere zatvoria, LED dióda zhasne.

- Po zistení dotyku na kapacitnom dotykovom senzore stiahneme aktuálne počasie v zvolenej lokalite. Aktuálnu teplotu vypíšeme do konzoly (na sériovú linku).

Schéma zapojenia pre tento scenár vyzerá podľa obrázku na predchádzajúcej strane dole.

Upozornenie: Vo svojich zapojeniach **NIKDY** nezapájajte LED diódy bez ochranného rezistora!!!

Krok 2. Prvé kroky

Aby ste mohli používať editor Thonny na programovanie mikrokontrolérov *ESP32*, potrebujete si skontrolovať nastavenie interpretéra. To vykonáte v menu `Run` > `Interpreter`. Tu si zo zoznamu vyberte položku `MicroPython (ESP32)`. Ako posledné si nastavte port, na ktorom je vaše zariadenie pripojené. Tu môžete vybrať voľbu pre automatickú detekciu, ktorá tento problém vyrieši za vás.

Po pripojení zariadenia a správnej konfigurácii sa vám v okne *Shell* zobrazí REPL režim jazyka *MicroPython* na mikrokontroléri *ESP32*. Komunikácia ide cez sériovú linku.

Ak chcete vedieť, aké moduly máte k dispozícii, môžete si ich všetky nechať vypísať príkazom `help('modules')`:

```
>>> help('modules')
__main__        inisetup          ucollections      urandom
_boot           math              ucryptolib        ure
_onewire        micropython       uctypes           urequests
_thread         neopixel          uerrno            uselect
_uasyncio       network           uhashlib          usocket
_webrepl        ntptime           uheapq            ussl
apa106          onewire           uio               ustruct
btree           uarray            ujson             usys
builtins        uasyncio/__init__ umachine          utime
cmath           uasyncio/core     umqtt/robust      utimeq
dht             uasyncio/event    umqtt/simple      uwebsocket
esp             uasyncio/funcs    uos               uzlib
esp32           uasyncio/lock     upip              webrepl
flashbdev       uasyncio/stream   upip_utarfile     webrepl_setup
```

```
framebuf      ubinascii         uplatform
gc            ubluetooth        upysh
Plus any modules on the filesystem
```

Poznámka: V zozname modulov si môžete všimnúť, že mnohé z nich majú v názve prefix s písmenom `u` (symbol pre *mikro*). Častokrát sa jedná o štandardný modul jazyka *Python* upravený pre použitie v jazyku *MicroPython*. To napríklad znamená, že ak potrebujete pracovať s formátom *JSON* a ste zvyknutí na prácu so štandardným modulom `json`, v jazyku *MicroPython* máte k dispozícii jeho *"micro"* verziu s názvom `ujson`. Interpreter jazyka *MicroPython* tiež zvláda mnohé moduly importovať na základe ich názvov zo štandardnej knižnice. Takže miesto:

```
import ujson
```

môžete modul importovať aj

```
import json
```

Krok 3. Blikanie LED diódou v režime REPL

LED diódu máme pripojenú ku pin-u s číslom *21*. Budeme ju používať na signalizáciu stavu dverí, čo znamená, že LED dióda bude svietiť vtedy, keď budú dvere otvorené a diódu zhasneme, keď sa dvere zatvoria. Pri predstavení práce s LED diódou si ukážeme silu, ktorú nám ponúka *REPL* režim jazyka *MicroPython*.

Začneme tým, že z balíka `machine` importujeme triedu `Pin`:

```
>>> from machine import Pin
```

Vytvoríme objekt triedy `Pin`, ktorý bude reprezentovať *LED* diódu. Konštruktor triedy `Pin` má tieto parametre:

- `id` – povinný parameter, ktorý identifikuje pin,
- `mode` – režim pin-u, ktorý môže byť:
  - `Pin.IN` – pin pre vstup, a,
  - `Pin.OUT` – pin pre výstup,
- `pull` – špecifikuje, či má mať pin pripojený (slabý) pull rezistor a jeho hodnota môže byť:
  - `Pin.PULL_UP` – zapnutý pull up rezistor,
  - `Pin.PULL_DOWN` – zapnutý pull down rezistor,
  - `None` – žiadny pull up alebo pull down rezistor.

Poznámka: Trieda `Pin` má výrazne viac možností konfigurácie a práce. Pre podrobnejšie informácie sa pozrite do dokumentácie.

Výsledný kód pre vytvorenie výstupného pin-u na pin-e č. *21* s pripojeným pull-down rezistorom bude vyzerať nasledovne:

```
>>> led = Pin(21, Pin.OUT, Pin.PULL_DOWN)
```

Pre prácu s diódou, resp. všeobecne s digitálnym pin-om, máme k dispozícii niekoľko metód. Ak chceme získať aktuálny stav pin-u, zavoláme metódu `value()`:

```
>>> led.value()
0
```

Poznámka: Pin sme pri jeho vytváraní nijako neinicializovali, takže hodnota na pin-e v skutočnosti nemusí byť `0`.

Pomocou metódy `value()` však môžeme aj hodnotu na pin zapísať. Ak chceme *LED* diódu pripojenú na tento pin zasvietiť, do metódy `value()` zapíšeme ako parameter hodnotu `1`:

```
>>> led.value(1)
```

Ak následne chceme *LED* diódu zhasnúť, do metódy `value()` vložíme ako parameter hodnotu `0`:

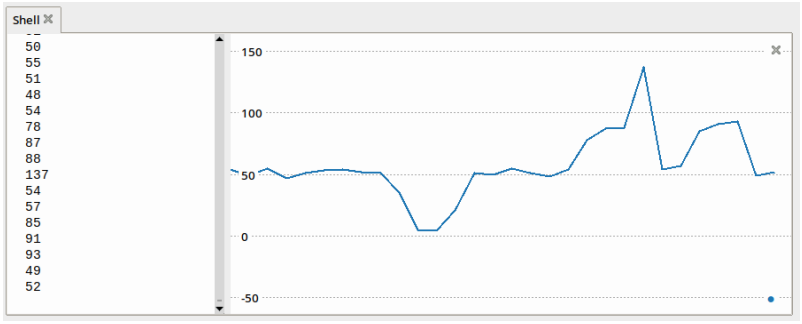
```
>>> led.value(0)
```

Trieda `Pin` však umožňuje nad svojimi inštanciami volať aj priamo metódu `on()` na zasvietenie (alternatíva pre `value(1)`) a metódu `off()` pre zhasnutie (alternatíva pre `value(0)`).

```
>>> led.on()
>>> led.off()
```

Na záver už len spojíme všetko dokopy a vytvoríme jednoduchý blikač, ktorý bude blikať v sekundových intervaloch. Kód môžete zapísať do súboru `playground.py` a následne ho spustiť:

```
from machine import Pin
from time import sleep
led = Pin(21, Pin.OUT)
while True:
    led.on()
    sleep(1)
    led.off()
    sleep(1)
```



Obr. 4: Vykreslenie priebehu hodnôt z hallovho senzora

Krok 4. Hallova sonda/senzor

Mikrokontrolér *ESP32* je vybavený *hallovým senzorom*, ktorý detekuje prítomnosť magnetického poľa. Jeho sila je vyjadrená hodnotou, ktorú tento senzor vráti – čím je pole väčšie/silnejšie, tým je aj hodnota senzora vyššia.

Silu magnetického senzoru v okolí mikrokontroléra *ESP32* prečítame zavolaním funkcie `hall_sensor()`, ktorá sa nachádza v module `esp32`:

```
>>> from esp32 import hall_sensor
>>> hall_sensor()
44
```

Platí, že ak v okolí Hallovho senzora nie je magnetické pole, bude vrátená hodnota nízka. Ak sa však v okolí bude magnetické pole nachádzať, senzor bude vracať vysoké hodnoty.

Správanie senzora si môžeme otestovať veľmi jednoducho – vytvoríme nekonečnú slučku a v pravidelných intervaloch budeme odčítavať hodnoty z *hallovho senzora*. Túto slučku môžeme uložiť do súboru `playground.py`, ktorý budeme použiť na experimenty a výsledok môžeme zobrazovať pomocou *plotter*-a.

```
from esp32 import hall_sensor
from time import sleep
while True:
    print(hall_sensor())
    sleep(0.5)
```

V našom scenári budeme *hallov senzor* používať ako senzor otvorených/zatvorených dverí. Pre tento účel máme v súbore `helpers.py` vytvorenú funkciu `is_door_open()`, ktorá nám na základe veľkosti magnetického poľa vráti aktuálny stav dverí:

- vráti hodnotu `True`, ak sú dvere otvorené (v tomto prípade senzor vráti nízku hodnotu), alebo,
- vráti hodnotu `False`, ak sú dvere zatvorené (v tomto prípade senzor vráti vysokú hodnotu).

Funkcia je veľmi jednoduchá a vyzerá takto:

```
from esp32 import hall_sensor
HALL_TRESHOLD = 150
def is_door_open():
    return hall_sensor() < HALL_TRESHOLD
```

Vyskúšať ju môžete priamo v REPL režime takto:

```
>>> from helpers import is_door_open
>>> is_door_open()
True
```

Poznámka: Hodnotu, pri ktorej budeme považovať dvere za zatvorené, si musí každý nastaviť sám, nakoľko citlivosť senzora a sila použitého magnetu môže byť na rozličných zariadeniach iná. Táto hodnota je uložená v premennej `HALL_TRESHOLD` v module `workshop.py`.

Krok 5. Superloop

Je načase dať veci dokopy a vytvoriť senzor otvorených dverí využívajúci *hallov senzor* a LED diódu. Aktualizujeme teda obsah súboru `workshop.py`.

```
from machine import Pin
from time import sleep
from helpers import is_door_open
if __name__ == '__main__':
    # init door
    door_state = is_door_open()
    # init led
    led = Pin(21, Pin.OUT, Pin.PULL_DOWN)
    led.value(door_state)
    while True:
        # check state of the door
        if door_state != is_door_open():
            door_state = not door_state  # is_door_open()
            led.value(door_state)
            if door_state == True:
                print('>> Door has been opened.')
```

```
            else:
                print('>> Door has been closed.')
        sleep(0.5)
```

Krok 6. Vnútorný senzor teploty

Mikrokontrolér *ESP32* obsahuje aj vnútorný senzor teploty. Ten síce nemeria vonkajšiu teplotu okolia, ale pracovnú teplotu mikrokontroléra. Za istých okolností ho je však možné použiť aj ako senzor teploty prostredia. To je však možné len v tom prípade, ak okolitá teplota je vyššia ako pracovná teplota mikrokontroléra. A tá je v prípade mikrokontroléra *ESP32* častokrát viac ako *40°C*.

Aktuálnu pracovnú teplotu mikrokontroléra vieme odmerať zavolaním funkcie `raw_temperature()`, ktorá sa nachádza v module `esp32`:

```
>>> from esp32 import raw_temperature
>>> raw_temperature()
117
```

Hodnota, ktorú funkcia vráti je vo *Fahrenheitoch*. Ak ju chceme v stupňoch celzia, musíme ju skonvertovať podľa vzťahu:

```
temp_c = (value - 32.0) / 1.8
```

Aj hodnoty teplomera môžeme vizualizovať pomocou plotter-a vypisovaním nameraných hodnôt v nekonečnej slučke. Nakoľko však pracovná teplota bude prevyšovať *40°C*, jej zmenu budeme dosahovať ťažšie. Mikrokontrolér napr. môžete skúsiť priložiť ku vetráku vášho laptopu.

Pre potreby nášho scenára sa v module `helpers.py` nachádza funkcia `get_temperature()`, ktorá nám vždy vráti hodnotu teploty už prevedenú na stupne celzia. Funkcia je veľmi jednoduchá a vyzerá takto:

```
from esp32 import raw_temperature
def get_temperature():
    return (raw_temperature() - 32.0) / 1.8
```

Jej správanie si môžete overiť priamo v REPL režime napríklad takto:

```
>>> from helpers import get_temperature
>>> get_temperature()
46.66667
```

Krok 7. Superloop Update I.

Našu hlavnú slučku môžeme aktualizovať vypísaním pracovnej teploty pri každej aktualizácii:

```
while True:
    # ...
    print(f'{get_temperature()}°C')
    sleep(0.5)
```

Krok 8. Kapacitný senzor dotyku

Mikrokontrolér *ESP32* obsahuje *10* kapacitných dotykových GPIO pinov. Konkrétne sa jedná o piny `0`, `2`, `4`, `12`, `13`, `14`, `15`, `27`, `32` a `33` (na obrázku s rozložením pin-ov sú označené ako `TOUCH0` až `TOUCH9`). Tieto piny dokážu detekovať zmeny vo všetkom, čo obsahuje elektrický náboj. Napríklad aj v ľudskej koži. To znamená, že tieto piny dokážu detekovať to, keď sa ich dotýkate napríklad prstom.

Ak chceme pracovať s kapacitným senzorom dotyku na týchto pin-och, musíme vytvoriť inštanciu triedy `TouchPad`, ktorá sa nachádza v module `machine`. Jej jediným parametrom je objekt typu `Pin`, ktorým identifikujeme konkrétny pin. Zmenu kapacity následne zistíme zavolaním metódy `.read()` nad vytvoreným objektom typu `TouchPad`:

```
>>> from machine import Pin, TouchPad
>>> tp = TouchPad(Pin(14))
>>> tp.read()
647
```

V prípade, že bol detekovaný dotyk, hodnota, ktorú senzor vráti, bude nízka (rádovo jednotky až desiatky). V prípade, že dotyk detekovaný nebol, hodnota senzora bude vysoká (rádovo stovky). To si môžeme demonštrovať jednoduchou nekonečnou slučkou v súbore `playground.py`:

```
from machine import Pin, TouchPad
from time import sleep
tp = TouchPad(Pin(14))
while True:
    print(tp.read())
    sleep(0.5)
```

Pre potrebu nášho scenára sa v module `helpers.py` nachádza funkcia `was_touch()`, ktorá vráti hodnotu `True`, ak bol detekovaný dotyk na zvolenom pin-e. V opačnom prípade vráti hodnotu `False`. Parametrom tejto funkcie je číslo pin-u, na ktorom chcete detekovať dotyk (v našom prípade je to pin č. `14`). Funkcia je veľmi jednoduchá a jej kód vyzerá takto:

```
from machine import TouchPad, Pin
TOUCH_TRESHOLD = 50
```

```
def was_touch(pin):
    touchpad = TouchPad(Pin(pin))
    return touchpad.read() < TOUCH_TRESHOLD
```

Správanie funkcie si môžete overiť priamo v REPL režime napríklad takto:

```
>>> from helpers import was_touch(pin)
>>> was_touch(14)
False
```

Poznámka: Podobne ako v prípade *hallovho senzora*, aj tu sa môžu hodnoty vrátené metódou `.read()` líšiť. Preto vašu hraničnú hodnotu si môžete nastaviť v premennej `TOUCH_TRESHOLD` v module `workshop.py`.

Krok 9. Superloop Update II.

Po detekovaní dotyku v našej výslednej aplikácii dôjde k stiahnutiu dát o počasí z internetu. Keďže ale zatiaľ nie sme pripojení na internet, miesto počasia len vypíšeme na obrazovku, že sme detekovali dotyk. Neskôr túto časť nahradíme informáciami o počasí.

Po úpravách bude náš kód v súbore `workshop.py` vyzerať takto:

```
from esp32 import hall_sensor
from machine import Pin
from time import sleep
from helpers import is_door_open, was_touch, get_temperature

if __name__ == '__main__':
    # init door
    door_state = is_door_open()
    # init led
    led = Pin(21, Pin.OUT, Pin.PULL_DOWN)
    led.value(door_state)
    # init touchpad
    touch_state = was_touch(14)
    while True:
        # check state of the door
        if door_state != is_door_open():
            door_state = not door_state  # is_door_open()
            led.value(door_state)
            if door_state == True:
                print('>> Door has been opened.')
            else:
```

```
            print('>> Door has been closed.')
        # check the state of touch pad
        if touch_state != was_touch(14):
            touch_state = not touch_state  # was_touch(tp)
            if touch_state is True:
                print('>> Touch detected.')
        # print temperature
        print(f'{get_temperature()}°C')
        sleep(0.5)
```

Krok 8. Pripojenie do siete WiFi

Pre pripojenie k WiFi použijeme modifikovanú verziu funkcie `do_connect()`, ktorá sa nachádza v module `helpers.py`. Túto funkciu odporúčajú použiť aj autori dokumentácie jazyka *MicroPython* pre mikrokontrolér *ESP32*. Jej pôvodnú verziu môžete nájsť na tejto adrese. Zmeny sú len mierne:

- funkcia má parameter pre SSID a heslo do WiFi siete,
- po úspešnom pripojení dôjde ku synchronizácii hodín pomocou protokolu NTP.

```
def do_connect(ssid, password):
    import network, ntptime, machine
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    if not wlan.isconnected():
        print('>> Connecting to network...')
        wlan.connect(ssid, password)
        while not wlan.isconnected():
            pass
    print('>> Network config:', wlan.ifconfig())
    # set time and date with NTP
    print('>> Synchronizing time...')
    ntptime.settime()
    rtc = machine.RTC()
    now = rtc.datetime()
    print(f'>> Current time: {now[0]}-{now[1]:02}-{now[2]:02}
        T{now[4]:02}:{now[5]:02}:{now[6]:02}Z')
```

Funkciu môžeme samozrejme otestovať priamo v REPL režime:

```
>>> from helpers import do_connect
>>> do_connect('ssid', 'password')
```

Ak máme správny názov siete a rovnako tak máme prístupové heslo do siete, po úspešnom pripojení sa zobrazia tieto informácie:

- pridelená IP adresa, maska siete a brána,
- aktuálny čas (UTC), ktorý sa získa z NTP servera.

```
>>> from helpers import do_connect
>>> do_connect('ssid', 'password')
>> Network config: ('10.20.30.144', '255.255.255.0',
    '10.20.30.1', '10.20.30.1')
>> Synchronizing time...
>> Current time: 2022-09-08T09:14:15Z
```

Funkciu budeme volať v module `workshop.py` ako prvú po spustení – ak nebudeme pripojení do internetu, náš scenár nebude fungovať.

```
if __name__ == '__main__':
    do_connect('ssid', 'password')
    # ...
```

V tomto momente sme úspešne pripojení s našim mikrokontrolérom do WiFi siete a môžeme sa pozrieť na možnosti jeho sieťovej komunikácie.

Krok 9. MicroPython a komunikačný protokol MQTT

Priamo vo firmvéri *MicroPython*-u, ktorý je nahratý v mikrokontroléri, sa nachádza podpora pre komunikačný protokol *MQTT*. Jedná sa o moduly `umqtt.simple` a `umqtt.robust`. Budeme používať modul `umqtt.robust` a pripájať sa budeme k verejnému MQTT brokeru na adrese `broker.hivemq.com` na štandardnom porte pre *MQTT* protokol `1883`.

Začneme tým, že si vytvoríme objekt klienta. Pre jeho vytvorenie budeme potrebovať:

- jednoznačný identifikátor klienta; očakáva sa reťazec,
- adresu MQTT brokera; v našom prípade použijeme adresu `broker.hivemq.com`,
- komunikačný port; predvolená hodnota je štandardné číslo portu `1883`, takže ho nastavovať nemusíme.

Pripojiť sa k broker-u môžeme priamo z REPL režimu mikrokontroléra volaním:

```
>>> from umqtt.robust import MQTTClient
>>> client = MQTTClient('jedinecne-id-klienta',
    'broker.hivemq.com')
>>> client.connect()
```

Poznámka: Pre potreby workshop-u stačí, ak ako identifikátor použije každý účastník svoje meno. Problém však môže nastať, ak si dvaja účastníci zvolia rovnaký identifikátor. Pri reálnych riešeniach je však potrebné zabezpečiť, aby ste nemuseli zadávať identifikátor ručne a aj napriek tomu aby bola zabezpečená jeho jedinečnosť. Za týmto účelom môžete použiť volanie `ubinascii.hexlify(machine.unique_id())`, ktoré prekonvertuje identifikátor zariadenia (získaný volaním `machine.unique_id()`) na reťazec.

Na otestovanie spojenia môžeme začať posielať správy z mikrokontroléra do dohodnutej témy, ktorou môže byť napríklad:

`pycon/messages`

Priamo z REPL režimu môžeme správu do uvedenej témy poslať takto:

```
>>> client.publish('pycon/messages',
    'pozdrav z mikrokontrolera esp32')
```

Aby sme si overili, že tieto správy do témy `pycon/messages` naozaj odchádzajú, môžeme sa prihlásiť na odber správ doručovaných do tejto témy. To môžeme zabezpečiť viacerými spôsobmi:

- pomocou HiveMQ webového klienta rovno v prehliadači,
- pomocou niektorého desktopového klienta, alebo,
- môžeme použiť nástroje z príkazového riadku v prostredí OS Linux z balíčka `mosquitto-clients`
  ```
  $ mosquitto_sub -h broker.hivemq.com
  -F '@Y-@m-@dT@H:@M:@S@z : %p' -t 'pycon/messages'
  ```

Krok 10. Publikovanie teploty cez protokol MQTT

V našom scenári budeme pomocou protokolu *MQTT* publikovať nameranú teplotu. Aktualizujeme teda náš program o pripojenie k MQTT brokerovi a o publikovanie údajov o teplote do kanála

`pycon/sk/2022/meno_pouzivatela/door`

MQTT klienta vytvoríme rovno po pripojení k sieti:

```
if __name__ == '__main__':
    # connect to wifi
    do_connect('ssid', 'password')
    # connecting to mqtt broker
    client = MQTTClient('id-klienta', 'broker.hivemq.com')
    client.connect()
    # ...
```

Publikovať teplotu budeme na/za riadkom, kde ju vypisujeme pomocou funkcie `print()`. Dôležité je však uviesť, že publikovaná hodnota/správa musí byť typu reťazec (`str`). Preto je potrebné pred odoslaním dáta najprv na reťazec konvertovať:

```
# ...
# publish temperature
print(f'{get_temperature()}°C')
client.publish('pycon/sk/2022/mirek/temp',str(get_temperature()))
sleep(0.5)
```

Správy môžeme sledovať ako pri testovaní – napríklad z príkazového riadku pomocou nástroja `mosquitto_sub`:

```
$ mosquitto_sub -h broker.hivemq.com -F '%t %p'
    -t 'pycon/sk/2022/mirek/temp'
```

Podobným spôsobom môžeme publikovať aj údaj o stave dverí. Do časti programu, kde ošetrujeme zmenu stavu dverí môžeme pripísať riadok:

```
client.publish('pycon/sk/2022/mirek/door',str(int(door_state)))
```

Krok 11. Stiahnutie informácií o počasí cez protokol HTTP

Podľa scenára máme v prípade detekovania dotyku stiahnuť z internetu o počasí pre zadanú lokáciu. Využijeme na to REST API služby openweathermap.org, s ktorou budeme komunikovať pomocou komunikačného protokolu HTTP. Na tento účel použijeme modul `urequests`, ktorý je mikro verziou známeho a populárneho balíčka v Pythone s názvom `requests`.

Dopyt pre získanie aktuálneho počasia v *Bratislave* v režime REPL bude vyzerať takto:

```
>>> import urequests
>>> url='http://api.openweathermap.org/data/2.5/weather?units=
    metric&q=bratislava&appid=9e547051a2a00f2bf3e17a160063002d'
>>> response = urequests.get(url)
```

Poznámka: Kľúč pre prístup k REST API v požiadavke uvedený ako parameter `appid` je vytvorený len pre potreby tohto workshop-u. Po jeho skončení nebude fungovať. Pre získanie vlastného kľúča sa zdarma zaregistrujte na uvedenej službe.

Následne sa môžeme pozrieť na výsledok nášho dopytu. HTTP stavový kód zobrazíme zapísaním:

```
>>> response.status_code
200
```

Dáta, ktoré sme získali pomocou dopytu, sa nachádzajú vo formáte JSON. Tie vieme skonvertovať na slovník zavolaním:

```
>>> data = response.json()
```

Ich zobrazením uvidíme všetky údaje, ktoré sme získali zo služby:

```
>>> print(data)
```

Pre zjednodušenie práce je v module `helpers.py` vytvorená funkcia nazvaná `get_current_weather()`, ktorá pre zvolenú lokáciu vráti slovník len s názvom lokácie, kódom krajiny a aktuálnou teplotou. Funkcia vyzerá takto:

```
import urequests
APPID = '9e547051a2a00f2bf3e17a160063002d'
def get_current_weather(location):
    url=f'http://api.openweathermap.org/data/2.5/weather?units=
      metric&q={location}&appid={APPID}'
    response = urequests.get(url)
    data = response.json()
    response.close()
    return {
        'location': data['name'],
        'country': data['sys']['country'],
        'temp': data['main']['temp']
    }
```

Funkcia sa dá vyskúšať jednoducho z REPL režimu takto:

```
>>> from helpers import get_current_weather
>>> get_current_weather('kosice')
{'country': 'SK', 'temp': 17.25, 'location': 'Kosice'}
```

Krok 12. Prezentovanie informácií o počasí po detekovaní dotyku

V module `workshop.py` nájdeme miesto, kde po detekcii dotyku vypisujeme do konzoly text. Za funkciu `print()` pridáme ďalšie riadky kódu:

```
if touch_state is True:
    print('>> Touch detected.')
    data = get_current_weather('bratislava')
    print(f'>> Aktuálna teplota v meste {data["location"]}
      ({data["country"]}) je {data["temp"]}°C')
```

Obr. 5: ESP-NOW logo

Krok 13. ESP-NOW beacon

Maják (z angl. *beacon*) je (bezdrôtové) zariadenie, ktoré v pravidelných intervaloch vysiela rádiový signál s údajmi pre iné zariadenia. No a takýto maják s údajmi, ktoré máme, si vyrobíme z mikrokontroléra *ESP32* a vysielať ho budeme pomocou komunikačného protokolu *ESP-NOW*.

O protokole *ESP-NOW*

ESP-NOW je bezdrôtový komunikačný protokol, ktorý podporuje:

- priame spojenie s max. 20 zaregistrovanými zariadeniami bez WiFi protokolu,
- šifrovanú a otvorenú komunikáciu (naraz je možné šifrovane komunikovať s max. 6 klientmi),
- max. dĺžku správy *250* bytov,
- súčasnú komunikáciu aj s WiFi na mikrokontroléroch *ESP32* a *ESP8266*.

Tento protokol je vhodný na riešenia v malých *IoT* sieťach, v riešeniach citlivých na oneskorenie alebo na nízku spotrebu, prípadne v riešeniach pre komunikáciu na veľké vzdialenosti medzi zariadeniami (až stovky metrov).

Tento protokol taktiež umožňuje sledovať silu WiFi signálu (RSSI) komunikujúcich zariadení.

Inicializácia vysielania

Pre prácu s protokolom *ESP-NOW* musíme mať zapnutý a aktivovaný *WiFi* modul. To znamená, že potrebujeme v našom kóde spustiť aspoň tieto riadky:

```
import network
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
```

V našom prípade túto aktiváciu rieši funkcia `do_connect()`, takže nepotrebujeme náš kód nijako upravovať. Ak však budete riešiť aplikáciu, ktorá bude komunikovať len pomocou tohto protokolu, vaše riešenie musí tieto riadky obsahovať.

Okrem toho však treba vytvoriť objekt z triedy `ESPNow`. To urobíme pomocou nasledujúcich riadkov:

```
from espnow import ESPNow
espnow = ESPNow()
espnow.active(True)
```

Manažment peer-ov

Predtým, ako budeme môcť poslať správu inému zariadeniu (z angl. *peer*), musíme toto zariadenie zaregistrovať. Zaregistrovať zariadenie je možné zavolaním metódy `.add_peer()` nad objektom typu `ESPNow` s MAC adresou zariadenia, ktorému chceme posielať správu.

MAC adresu je možné zistiť na cieľovom zariadení nasledovne:

```
>>> import network
>>> wlan = network.WLAN(network.STA_IF)
>>> wlan.config('mac')
b'\xb4\xe6-\x9e7M'
```

Zistené zariadenie nasledovne vieme zaregistrovať takto:

```
>>> espnow.add_peer(b'\xb4\xe6-\x9e7M')
```

Overiť registráciu môžeme napríklad vypísaním zoznamu všetkých zaregistrovaných zariadení volaním:

```
>>> espnow.get_peers()
((b'\xb4\xe6-\x9e7M', b'\x00\x00\x00\x00\x00\x00\x00\x00
    \x00\x00\x00\x00\x00\x00\x00\x00', 0, 0, False),)
```

Príprava správy

Správa, ktorú budú naše majáky vysielať, musí mať max. *250* znakov. Preto ju budeme vysielať ako jednoduchý reťazec v *CSV* formáte. Správa bude v tvare:

```
meno majáka; teplota; úroveň mag. poľa
```

Správu si pripravíme do premennej `payload` takto:

```
>>> payload = '{};{};{}'.format(
    'makac',
    get_temperature(),
    hall_sensor())
```

Odoslanie správy

Pripravenú správu odošleme zavolaním metódy `.send()` nad objektom typu `ESPNow`. Táto metóda má dva parametre:

- MAC adresa zariadenia, ktorému správu posielame, a,
- samotnú správu.

Odoslanie pripravenej správy bude teda vyzerať nasledovne:

```
>>> espnow.send(b'\xb4\xe6-\x9e7M', payload)
```

Ďalšie zdroje

- Namakaný deň: ESP32 labs – ďalšie laby pre mikrokontrolér *ESP32*, ktoré predstavujú pripojenie a prácu s niektorými senzormi a akčnými členmi.
- MicroPython – Domovská stránka projektu *MicroPython*.
- Quick reference for the ESP32 – Skrátená dokumentácia jazyka *MicroPython* pre dosku s mikrokontrolérom *ESP32*.
- Random Nerd Tutorials – Portál venovaný nie len programovaniu mikrokontroléra *ESP32* v jazyku *MicroPython*.
- ESP32 Labs – Niekoľko labov pre začiatočníkov v jazyku *MicroPython* s mikrokontrolérom *ESP32*.
- Last Minute Engineers: ESP32 Projects – Časť portálu Last Minute Engineers venovaná konkrétne projektom a informáciám o mikrokontroléri *ESP32*.
- HiveMQ Web Client – MQTT klient vo webovom prehliadači.
- Support for the ESP-NOW protocol – dokumentácia pre komunikačný protokol *ESP-NOW* v jazyku *MicroPython*

Licencia

PRÁCA S PROTOKOLMI HTTP A MQTT V JAZYKU MICROPYTHON

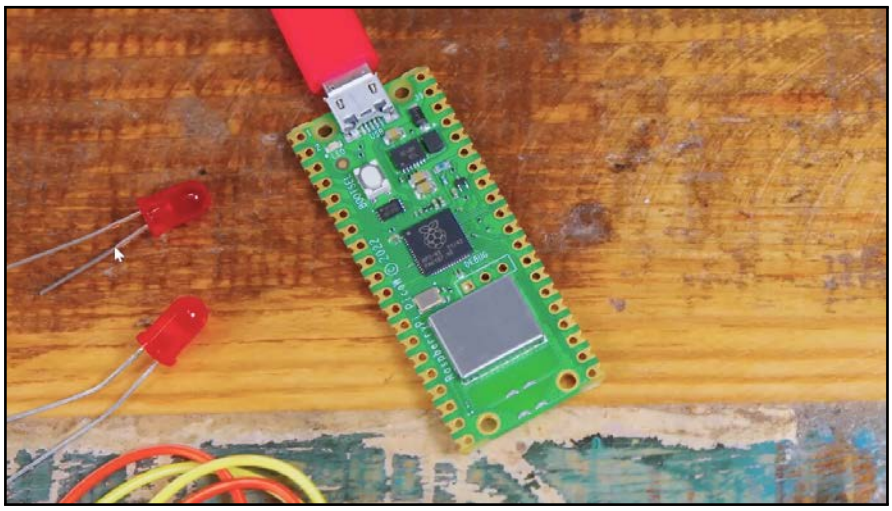
WORKING WITH HTTP AND MQTT PROTOCOLS IN MICROPYTHON

Mirek Biňas

E-mail: miroslav.binas@tuke.sk

Doska *Raspberry Pi Pico* patrí do rodiny zariadení vyrobených nadáciou *Raspberry Pi Foundation*. Nejedná sa však o minipočítač, ako je to v prípade minipočítačov *Raspberry Pi*, ale jedná sa o dosku osadenú mikrokontrolérom *RP2040*. Vďaka cene, svojim vlastnostiam, komunite a podpore nadácie Raspberry Pi Foundation, je doska *Raspberry Pi Pico* ideálnou pomôckou na výučbu programovania mikrokontrolérov. A počas tejto tvorivej dielne si predstavíme základy práce s touto doskou v jazyku *MicroPython*. Vytvoríme jednoduché riešenie, dosku pripojíme do internetu, potom pomocou HTTP protokolu stiahneme informácie o počasí, aby sme ich následne uložili do *Adafruit IO* pomocou protokolu HTTP aj MQTT.

Odporúčaný čas: 180 minút.



Obr. 1: Raspberry Pi Pico W

Ciele

1. Naučiť sa nainštalovať firmvér s jazykom *MicroPython*.
2. Naučiť sa základy práce v REPL režime jazyka *MicroPython*.
3. Naučiť sa pripojiť dosku *Raspberry Pi Pico W* do internetu pomocou zabudovaného *WiFi* modulu.
4. Naučiť sa publikovať údaje z mikrokontroléra pomocou komunikačného protokolu *MQTT*.
5. Naučiť sa základy používania modulu `urequests` pre prácu s HTTP protokolom.

Čo budeme robiť

Na tomto workshope si vyskúšame základy práce s mikrokontrolérom *Raspberry Pi Pico W*, kde si ukážeme práve to, ako je možné komunikovať pomocou HTTP a MQTT protokolu. Ukážeme si, ako do neho nainštalovať firmvér, ako sa priprojiť na WiFi sieť, stiahnuť dáta o aktuálnom počasí, a pretransformujeme ich do formy, ktorú potrebuje služba Adafruit IO.

Vytvoríme vlastne jednoduché ETL, ktoré bude vyzerať takto:

```
[ extract_data ] --> [ transform_data ] --> [ load_data ]
```

Každú jednu úlohu pritom implementujeme vo forme samostatnej Python funkcie.

Naivné riešenie

Vytvárať budeme veľmi naivné riešenie, kde sa nebudeme venovať mnohým kontrolám, ktoré by sme pri reálnom nasadení riešili, ako napr. HTTP status odpovedí, výpadok spojenia a pod. Niekoľko odporúčaní sa tu ale objaví.

Niekoľko tipov pre prácu s mikrokontrolérom

- Pico na doske neobsahuje tlačidlo `RESET`. Reštartovať ho je však jednoduché odpojením a opätovným pripojením USB kábla. **Vyhnite sa však jeho odpájaniu a pripájaniu na strane mikrokontroléra!** Vyhnete sa tak možnosti odtrhnutia tohto konektoru a tým pádom aj jeho celkovému poškodeniu. Miesto toho odpájajte USB kábel na strane počítača.
- V prípade, že chcete mikrokontrolér reštartovať softvérovo, môžete tak urobiť príkazom:

```
from machine import reset; reset()
```

- Softvérový reset zariadenia sa dá vykonať ešte jednoduchšie – stlačením klávesovej skratky `CTRL+D`.

Krok 0. Čo budeme potrebovať?

Ešte predtým, ako začneme, si pripravte nasledovné:

- Na svoj počítač nainštalujte jednoduchý editor kódu Thonny a v jeho menu `Zobraziť` zaškrtnite voľby `Shell` a `Súbory`.
- Ak pracujete v *OS Linux*, tak sa uistite, že váš používateľ má práva na prístup k zariadeniu (obyčajne je to skupina `dialout`). Ak to tak nie je, pridajte používateľa do uvedenej skupiny napr. týmto príkazom:

  ```
  $ sudo usermod -aG dialout USERNAME
  ```

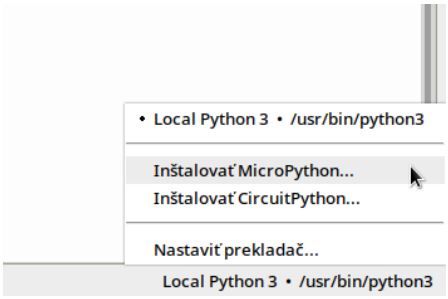
- Ak pracujete v *OS Windows*, uistite sa, že máte správne ovládače, a že systém mikrokontrolér rozpozná.
- Vytvorte si bezplatný účet v službe Open Weather.
- Vytvorte si bezplatný účet v službe Adafruit IO.

Krok 1. Nahratie firmvéru do mikrokontroléra

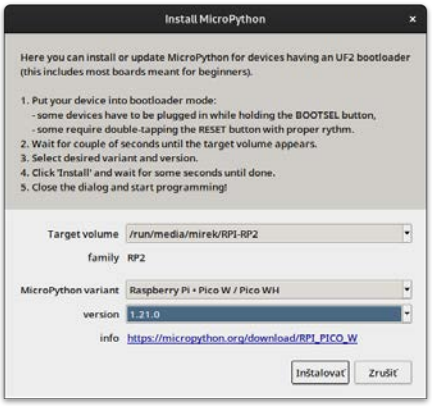
Ak chceme mikrokontrolér programovať v jazyku *MicroPython*, potrebujeme najprv do neho nahrať firmvér s týmto jazykom. S tým nám pomôže editor *Thonny*.

Pre nahratie firmvéru postupujte podľa tohto návodu:

1. Spustite editor *Thonny*.
2. Ak máte dosku *Raspberry Pi Pico* pripojenú k počítaču, tak ju odpojte. Na doske stlačte a držte tlačidlo `BOOTSEL` a dosku pripojte k počítaču.
3. Po pripojení dosky tlačidlo pustite. Váš operačný systém môže zobraziť notifikáciu o tom, že k vášmu počítaču bol pripojený USB disk s názvom `RPI-RP2`.
4. V pravom dolnom rohu editora *Thonny* kliknite na názov používaného interpretera jazyka *Python* a zo zoznamu možností vyberte položku `Inštalovať MicroPython...`.
5. V dialógovom okne `Inštalovať MicroPython` vyberte umiestenie pripojenej dosky Raspberry Pi Pico, vyberte variant dosky a verziu jazyka MicroPython, ktorú chcete nainštalovať.
6. Kliknite na tlačidlo `Inštalovať`, čím sa spustí inštalácia.



Obr. 2: Editor Thonny: Výber inštalácie jazyka MicroPython



Obr. 3: Editor Thonny: Dialóg pre inštaláciu jazyka MicroPython

7. Po skončení inštalácie zatvorte dialógové okno a v pravom dolnom rohu editora vyberte zo zoznamu vaše zariadenie s jazykom *MicroPython*, napr. `MicroPython (Raspberry Pi Pico)`.

Ak ste postupovali správne, v príkazovom riadku editora *Thonny* by ste mali vidieť prompt jazyka *MicroPython*, napr.:

```
MicroPython v1.21.0 on 2023-10-06; Raspberry Pi Pico W with RP2040
Type "help()" for more information.
>>>
```

Krok 2. Režim REPL a Hello world!

Jediný program, ktorý je v mikrokontroléri nahratý a spustený, je interpreter jazyka *MicroPython*. Komunikovať s ním je možné pomocou sériovej linky, kde uvidíme jeho rozhranie v režime REPL. To môžeme otestovať vypísaním textu `Hello world!` pomocou funkcie `print()`:

```
>>> print('Hello world!')
Hello world!
>>>
```

Podobne, ako v prípade štandardného jazyka *Python* môžeme režim REPL využiť na ladenie a experimentovanie.

Krok 3. Pripojenie k internetu

Ako základ pre pripojenie do internetu použijeme funkciu z dokumentácie jazyka MicroPython, ktorá vyzerá takto:

```
def do_connect():
    import network
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    if not wlan.isconnected():
        print('connecting to network...')
        wlan.connect('ssid', 'key')
        while not wlan.isconnected():
            pass
    print('network config:', wlan.ifconfig())
```

Mierne ju upravíme, aby jej parametrom bolo SSID pre pripojenie do siete a rovnako tak aj heslo:

```
def do_connect(ssid, password):
    import network
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    if not wlan.isconnected():
        print(f'>> Connecting to network "{ssid}"...')
        wlan.connect(ssid, password)
        while not wlan.isconnected():
            pass
    print('>> Network config:', wlan.ifconfig())
```

Funkciu môžeme rovno otestovať v *REPL* režime. Ak sa napríklad chceme pripojiť do siete `labak` s heslom `jahodka`, tak to urobíme takto:

```
do_connect('labak','jahodka')
```

Odporúčanie

Po vykonaní práce sa nezabudnite od siete odpojiť! To je dôležité kvôli zníženiu spotreby a uvoľneniu prostriedkov aj pre iných.

Odpojiť sa je možné pomocou dvoch metód nad sieťovým rozhraním:

1. `.disconnect()` – odpojí sa od siete, ale WiFi modul zostane aktívny a tým pádom spotreba bude vyššia,
2. `.deinit()` – okrem odpojenia aj vypne WiFi modul, čím dôjde aj k zníženiu spotreby zariadenia.

Krok 4. Stiahnutie aktuálneho počasia

Cieľom prvej úlohy v našom ETL je stiahnutie informácií o aktuálnom počasí. Vytvoríme preto funkciu `extract_data()`, v ktorej túto úlohu implementujeme.

Na získanie informácií o aktuálnom počasí budeme používať službu Open Weather. Služba poskytuje REST API, pomocou ktorého je rozličné sa dopytovať a získavať či už aktuálne alebo historické dáta o počasí.

Pre použitie dát zo služby je potrebné mať účet. Aktuálne počasie budeme vedieť získať aj pomocou bezplatného účtu. Na to použijeme modul `requests`, ktorý je mikro implementáciou dobre známeho a veľmi populárneho *Python* modulu s rovnomenným názvom `requests`.

Formát URL adresy

Obecne sa URL adresa skladá z niekoľkých častí:

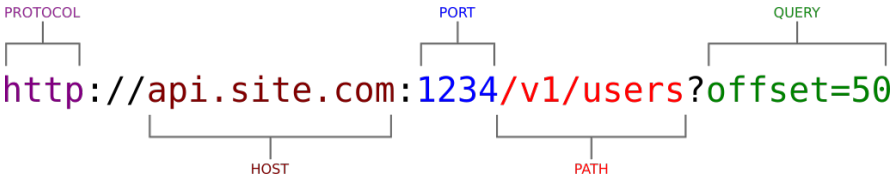
- schéme alebo protokol,
- host,
- port,
- cesta,
- parametre požiadavky.

Jednotlivé jej časti je možné vidieť na nasledovnom obrázku.

HTTP API

Dokumentácia pre HTTP požiadavky na získanie aktuálneho počasia sa nachádza na tejto stránke.

Ak by sme napríklad chceli získať informácie o aktuálnom počasí v Žiline v metrickej sústave, jej URL adresa bude vyzerať takto:



Obr. 4: Formát URL adresy

```
https://api.openweathermap.org/data/2.5/weather?q=zilina&
  units=metric&appid=9e547051a2a00f2bf3e17a160063002d
```

Samotnú HTTP požiadavku si môžeme overiť okrem prehliadača aj z príkazového riadku pomocou HTTP klientov, ako je napríklad `httpie` takto:

```
$ http https://api.openweathermap.org/data/2.5/weather \
  q==zilina \
  units==metric \
  appid==9e547051a2a00f2bf3e17a160063002d
```

HTTP požiadavka pomocou modulu `requests`

V jazyku MicroPython si takýto dopyt môžeme vyskúšať priamo v REPL režime takto:

```
>>> import requests
>>> response = requests.get('https://api.openweathermap.org/
    data/2.5/weather?q=zilina&units=metric&
    appid=9e547051a2a00f2bf3e17a160063002d')
```

Výsledkom bude objekt typu `Response`, ktorý má niekoľko metód. Keďže vieme, že odpoveď bude vo formáte JSON, môžeme nad objektom odpovede rovno zavolať metódu `.json()`:

```
>>> response.json()
```

Riešenie

Vytvoríme teda funkciu `extract_data()`, ktorá bude mať 3 parametre:

- `query` – požiadavku alebo mesto, o ktorom chceme získať informácie,
- `units` – v akých jednotkách bude výsledná hodnota (`metric` | `imperial` | `standard`),
- `appid` – aplikačný kľúč.

Funkcia vráti slovník s načítanými údajmi.
Riešenie môže vyzerať napríklad takto:

```
import requests
def extract_data(query: str, units: str, appid: str) -> dict:
    url = (
        'https://api.openweathermap.org/data/2.5/weather?'
        f'q={query}&units={units}&appid={appid}'
    )
    response = requests.get(url)
    data = response.json()
    response.close()
    return data
```

Krok 5. Spracovanie prijatých údajov

Našim cieľom je dostať získané dáta o počasí do služby Adafruit IO. Najprv sa však pozrieme na to, v akom formáte a v akom tvare služba od nás tieto dáta prijme a pred samotným odoslaním získané dáta do tohto tvaru transformujeme.

Adafruit IO kanály

Ešte skôr, ako začneme, potrebujeme sa zoznámiť s konceptov tzv. *kanálov* (z angl. *feeds*). V Adafruit IO sú kanály základným stavebným prvkom, pomocou ktorého sa ukladajú a organizujú dáta. Každý kanál je ako databázová tabuľka alebo kontajner, ktorý uchováva chronologický záznam hodnôt.

O kanáloch platí:

- Kanál je miesto, kam môžeš odosielať údaje (napr. teplota, vlhkosť, stav tlačidla atď.).
- Do kanála je možné údaje nielen zapisovať ale aj čítať.
- Používa sa na vizualizáciu údajov cez dashboardy (grafy, gauge, tlačidlá, prepínače).

- Každý kanál má svoje jedinečné meno a môže byť verejný alebo súkromný.
- Viacero kanálov môže byť organizovaných do jednej skupiny.

Pre naše potreby si teda vytvoríme skupinu `ossconf` a v nej si vytvoríme kanály pre merané veličiny `temp`, `humidity` a `pressure`.

Formát odosielaných dát

Dáta, ktoré budeme odosielať, musia byť v JSON formáte a budú obsahovať tieto kľúče:

- `lat` – zemepisná šírka,
- `lon` – zemepisná dĺžka,
- `value` – nameraná hodnota,
- `created_at` – čas merania v ISO formáte.

Takže JSON obsahujúci meranie v Žiline môže vyzerať takto:

```
{
    "lat": 49.2239,
    "lon": 18.7394,
    "value": 9.79,
    "created_at": "2025-07-02T06:23:00+02:00"
  }
```

Spracovanie údajov

Ešte predtým, ako sa však dostaneme ku samotnému odoslaniu dát do služby Adafruit IO v jazyku *MicroPython*, vytvoríme si dve pomocné funkcie.

Prvá funkcia zabezpečí jednoduchú konverziu časovej značky do ISO8601 formátu času.

```
def to_iso8601(ts: int) -> str:
    dt = time.gmtime(ts)
    return f'{dt[0]:04}-{dt[1]:02}-{dt[2]:02}
      T{dt[3]:02}:{dt[4]:02}:{dt[5]:02}Z'
```

Druhá funkcia bude reprezentovať samotnú úlohu nášho ETL pre transformáiu dát. Jej úlohou bude extrahovať len tie dáta, o ktoré máme záujem. To môže vyzerať napríklad takto:

```
def transform_data(data: dict) -> dict:
    return {
```

```
        'dt'       : to_iso8601(data['dt']),
        'lat'      : data['coord']['lat'],
        'lon'      : data['coord']['lon'],
        'metrics'  : {
            'temp'     : data['main']['temp'],
            'humidity' : data['main']['humidity'],
            'pressure' : data['main']['pressure']
        }
    }
```

Krok 6. Odoslanie údajov do Adafruit IO cez protokol HTTP

Odoslanie metriky cez protokol HTTP

Do každého kanálu môžeme odosielať údaje pomocou HTTP metódy `POST`. Každý kanál má pre tento účel k dispozícii konkrétnu URL adresu, ktorú získame po kliknutí na položku `Feed Info` v pravom bočnom paneli príslušného kanála. URL adresu nájdete pod kľúčom `API`.

Okrem toho sa však pri každej jednej požiadavke potrebujeme aj autentifikovať pomocou používateľského mena a kľúča. Tieto hodnoty získame po kliknutí na ikonku kľúča, kde tieto hodnoty získame, prípadne ich môžeme nechať znovu vytvoriť.

Overiť zápis do príslušného kanála z príkazového riadku môžeme nasledovne:

```
$ http post \
  https://io.adafruit.com/api/v2/{user}/
    feeds/{group}.{feed}/data \
  X-AIO-Key:{aio_key} \
  datum:='{
    "lat": 49.2239,
    "lon": 18.7394,
    "value": 9.79,
    "created_at": "2025-07-02T06:23:00+02:00"
  }'
```

Odoslanie metriky v jazyku *MicroPython*

V jazyku *MicroPython* v *REPL* režime to môžeme urobiť takto:

```
>>> import requests
>>> user = 'jano'
```

```
>>> group = 'ossconf'
>>> feed = 'temp'
>>> aio_key = 'secret.password'
>>> url = f'https://io.adafruit.com/api/v2/{aio_username}
      /feeds/{group}.{feed}/data'
>>> payload = {
    "lat": 49.2239,
    "lon": 18.7394,
    "value": 9.79,
    "created_at": "2025-07-02T06:23:00+02:00"
}
>>> headers = {'X-AIO-Key': aio_key}
>>> response=requests.post(url,headers=headers,json=payload)
```

Riešenie

Výsledná funkcia môže vyzerať napríklad takto:

```
def load_data(aio_username:str,aio_key:str,group:str,data:dict):
    headers = {'X-AIO-Key': aio_key}
    for feed in data['metrics']:
        url = f'https://io.adafruit.com/api/v2/{aio_username}
          /feeds/{group}.{feed}/data'
        payload = {
            'value': data['metrics'][feed],
            'lat': data['lat'],
            'lon': data['lon'],
            'created_at': data['dt']
        }
        response=requests.post(url,headers=headers,json=payload)
        response.close()
```

Krok 7. Odoslanie údajov do Adafruit IO cez protokol MQTT

Do kanálov je možné odosielať dáta aj pomocou protokolu MQTT. A presne to urobíme v tomto kroku.

Nainštalovanie balíka pre MQTT

Podpora protokolu MQTT vo firmvéri jazyka *MicroPython* nie je žiadna. Aby teda náš mikrokontrolér vedel komunikovať aj pomocou tohto protokolu, musíme do neho doinštalovať príslušný modul.

To vieme urobiť priamo z prostredia editora Thonny pomocou modulu pipkin. Problém však je, že od istej verzie jazyka *Python* už nepracuje správne.

Takže modul pre protokol MQTT nainštalujeme v *REPL* režime manuálne pomocou modulu `mip` nasledovne:

```
>>> import mip
>>> mip.install('umqtt.simple')
```

Po nainštalovaní tento modul uvidíte aj v súborovom systéme v priečinku `lib/`.

Pripojenie k MQTT brokeru

MQTT broker pre službu Adafruit IO sa nachádza na `io.adafruit.com` a komunikuje na štandardnom porte `1883`. Pri pripájaní použijete svoje používateľské meno a vygenerovaný kľúč, ku ktorému sa dostanete cez ikonu kľúčika (používali sme ho ako premennú `aio_key`).

Tému pre príslušný kanál zistíte opať v bočnom paneli kanála v položke `Feed Info`.

Odoslanie metriky cez protokol MQTT

To, či je všetko v poriadku, je možné otestovať z príkazového riadku napr. pomocou `mosquitto_pub` nástroja takto:

```
$ mosquitto_pub -h io.adafruit.com -u {user} -P {key} \
  -t {user}/feeds/{group}.{feed} \
  -m '{
    "lat": 49.2239,
    "lon": 18.7394,
    "value": 9.79,
    "created_at": "2025-07-02T06:23:00+02:00"
  }'
```

Odoslanie skupiny metrík cez protokol MQTT

Ak by ste chceli na jeden šup odoslať viacero metrík, je to možné dosiahnuť týmto spôsobom:

```
$ mosquitto_pub -h io.adafruit.com -u {user} -P {key} \
  -t {user}/groups/{group} \
  -m '{
    "feeds": {
      "pressure": 1016,
      "humidity": 94,
      "temp": 7.54
    },
    "location": {
      "lat": 48.6667,
      "lon": 21.3333
    }
  }'
```

Odoslanie údajov v jazyku MicroPython

Ak chceme odoslať meranie v jazyku *MicroPython*, môžeme si to vyskúšať v *REPL* režime pomocou týchto príkazov:

```
>>> user = 'jano'
>>> group = 'ossconf'
>>> feed = 'temp'
>>> aio_key = 'secret.password'
>>> topic = f'{aio_username}/groups/{group}'
>>> payload = {
    "lat": 49.2239,
    "lon": 18.7394,
    "value": 9.79,
    "created_at": "2025-07-02T06:23:00+02:00"
}
>>> from umqtt.simple import MQTTClient
>>> client = MQTTClient(unique_id(), 'io.adafruit.com', 1883,
                        aio_username, aio_key)
>>> client.connect()
>>> client.publish(topic, json.dumps(payload))
>>> client.disconnect()
```

Výsledné riešenie

Výsledná funkcia môže vyzerať napríklad takto:

```
from umqtt.simple import MQTTClient
def load_data_mqtt(aio_username: str, aio_key: str, group: str,
      data: dict):
    topic = f'{aio_username}/groups/{group}'
    payload = {
        "feeds": {
            "temp": data['metrics']['temp'],
            "humidity": data['metrics']['humidity'],
            "pressure": data['metrics']['pressure'],
        },
        "location": {
            'lat': data['lat'],
            'lon': data['lon'],
        }
    }
    client = MQTTClient(unique_id(), 'io.adafruit.com', 1883,
      aio_username, aio_key)
    client.connect()
    client.publish(topic, json.dumps(payload))
    client.disconnect()
```

Krok 8. Pravidelné spúšťanie

. . . v nekonečnej slučke

```
if __name__ == '__main__':
    while True:
        main()
        print(f'>> going to sleep for {INTERVAL} minutes')
        time.sleep(INTERVAL * 60)
```

. . . hlbokým uspaním

```
if __name__ == '__main__':
    main()
    print(f'>> going to sleep for {INTERVAL} minutes')
    machine.deepsleep(INTERVAL * 60 * 1000)
```

Bonus: Dekorátor osvietenia

```
from machine import Pin
def enlight(func):
    def wrapper(*args, **kwargs):
        led = Pin('LED', Pin.OUT)
        led.on()
        func(*args, **kwargs)
        led.off()
    return wrapper
```

Krok 9. Aktualizácia času

Počas pripájania k internetu vieme rovno synchronizovať aj čas pomocou NTP protokolu. Aktualizujeme a upravíme funkciu `do_connect()` nasledovne:

```
def do_connect(ssid, password):
    import network, ntptime, machine
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    if not wlan.isconnected():
        print('>> Connecting to network...')
        wlan.connect(ssid, password)
        while not wlan.isconnected():
            pass
    print('>> Network config:', wlan.ifconfig())
    # set time and date with NTP
    print('>> Synchronizing time...')
    ntptime.settime()
    rtc = machine.RTC()
    now = rtc.datetime()
    print(f'>> Current time: {now[0]}-{now[1]:02}-{now[2]:02}
      T{now[4]:02}:{now[5]:02}:{now[6]:02}Z')
```

Poznámka: Pri pripojení pomocou editora ako Thonny alebo nástroja `rshell`, dôjde k synchronizácii času automagicky. Počas reálnej prevádzky synchronizáciu musíme zabezpečiť sami.

Krok 10. Inštalácia doplnkových modulov

Pred verziou 1.20 sa používal modul `upip`, ktorý predstavoval MicroPython verziu nástroja `pip` pre inštaláciu balíkov. Od verzie 1.20 je však súča-

sťou firmvéru s MicroPython-om balík s názvom `mip`. Ten neinštaluje balíky z PyPi, ale z repozitára micropython-lib.

Nainštalovať nový modul môžeme priamo pomocou REPL. Najprv importneme modul `mip`:

```
>>> import mip
```

A následne nainštalujeme príslušný balík pomocou:

```
>>> mip.install('umqtt.simple')
```

Inštalované moduly sa inštalujú do priečinku `/lib/` v mikrokontroléri.

Ďalšie zdroje

- Raspberry Pi Pico and Pico W – technická dokumentácia pre rodinu mikrokontrolérov *Raspberry Pi Pico*,
- Package management – dokumentácia ku modulu `mip` pre správu balíčkov na mikrokontroléri s jazykom MicroPython,
- Propojka z Grove konektoru na 4 pin dupont samice,
- `https://wokwi.com/`,
- Raspberry Pi Pico W 101 – záznam z webinára.

Licencia

AKÝ JE TO POKÉMON?
WORKSHOP: WHO'S THAT POKÉMON?

Mirek Biňas

E-mail: miroslav.binas@tuke.sk

Aký je to Pokémon? alebo ako si vytvoriť Pokédex v jazyku Python pomocou webového rámca FastAPI.

Pokédex je pomôcka každého správneho trénera Pokémonov. Ak si trénerom Pokémonov, už určite svoj Pokédex máš. To ale nevadí, pretože na tomto workshope si vytvoríš svoj vlastný Pokédex, ktorý môže vyzerať a fungovať ako len chceš.

Pokédex vytvoríme pomocou mladého a stále populárnejšieho mikro webového rámca FastAPI. Keďže je primárne určený na tvorbu REST API, začneme endpoint-mi na získanie zoznamu všetkých Pokémonov, ale rovnako vytvoríme endpoint na získanie informácií o konkrétnom Pokémonovi na základe jeho čísla v Pokédexe. Ukážeme si však aj to, ako pomocou rámca FastAPI vytvárať dynamické HTML stránky pomocou šablónovacieho systému Jinja. A samozrejme – dáta o Pokémonoch sú uložené v SQLite databáze a budeme k nim pristupovať pomocou balíka SQLModel.

Obr. 1: Who's that Pokémon?

Odporúčaný čas: 120 až 180 minút.
Upozornenie: Keďže je tento workshop pomerne krátky, nebudeme v ňom aplikovať tie najlepšie prístupy pre tvorbu aplikácií pomocou rámca FastAPI.

Ciele

1. Naučiť sa základy práce s webovým rámcom FastAPI.
2. Naučiť sa vytvárať jednoduché modely pomocou rámca SQLModel.
3. Naučiť sa vytvárať dopyty pomocou ORM rámca SQLModel.
4. Naučiť sa vytvárať jednoduché REST API.
5. Naučiť sa vytvárať jednoduché HTML pohľady pomocou šablónovacieho systému Jinja.

Krok 0. Inštalácia prostredia

V tomto dokumente sa nachádzajú pokyny, ako sa na workshop pripraviť a čo si nainštalovať na svoj laptop ešte pred príchodom na workshop.

Pre potreby workshopu si prosím nainštalujte:

- jazyk Python, ideálne v najnovšej verzii (v dobe písania tohto návodu je aktuálna verzia 3.11), a,
- vývojové prostredie Visual Studio Code.

Nižšie nájdete pokyny pre inštaláciu pre váš operačný systém.

Inštalácia na Windows OS

Odporúčam vám inštalovať potrebný softvér pomocou balíčkovacieho systému Chocolatey:

- ako administrátor si nainštalujte balíčkovací systém Chocolatey podľa pokynov z tejto stránky,
- pre nainštalovanie IDE Visual Studio Code s interpreterom Python napíšte v príkazovom riadku (ako admin) tento príkaz:

```
$ choco install -y python3 vscode
```

Samozrejme si však oba balíčky môžete nainštalovať aj ručne:

- stiahnite si inštalátor jazyka Python pre svoj OS Windows z adresy `https://www.python.org/downloads/` a pri inštalácii **zaškrtnite** možnosť pridať záznam do premennej `PATH`,
- inštalátor prostredia Visual Studio Code stiahnete z domovskej stránky projektu `https://code.visualstudio.com/`.

Obr. 2: Visual Studio Code: Tlačidlo pre rozšírenia

Inštalácia na Linux OS

Jazyk Python sa už nachádza v OS Linux predinštalovaný. Overte si len to, či máte vhodnú verziu príkazom:

```
$ python3 --version
```

Vývojové prostredie Visual Studio Code môžete do svojej distribúcie nainštalovať aj pomocou balíčkov. Túto možnosť si ale overte vo svojej vlastnej distribúcii. V opačnom prípade na inštaláciu použite balíčky z domovskej stránky projektu.

Inštalácia rozšírení pre VS Code

Aby sme mohli pracovať vo VS Code v jazyku Python, potrebujeme ešte nainštalovať niekoľko rozšírení. Tie nainštalujete kliknutím na tlačidlo `Extensions`:

Následne vyhľadajte a nainštalujte tieto rozšírenia:

- Pylance,
- Python Indent,
- EditorConfig for VS Code.

Inštalácia projektu na OS Windows

1. Zo stránky `https://github.com/namakanyden/workshop-pokedex` si tiahnite priečinok `app-template/`. Ideálne si ho u seba uložte napr. na plochu, aby ste ho nemali ďaleko. Priečinok u seba premenujte napr. na `pokedex`.
2. Stlačte nad priečinkom `pokedex` pravé tlačidlo myši a vyberte položku `Open with Code`. Tým sa vám rovno spustí prostredie Visual Studio Code s uvedeným priečinkom. Pri otváraní už len potvrďte, že dôverujete obsahu priečinka.
3. V prostredí Visual Studio Code spustite príkazový riadok (menu `Terminal` > `New Terminal`). V spodnej časti obrazovky sa vám spustí terminál.

4. V príkazovom riadku zadajte nasledujúci príkaz:

   ```
   $ python3 -m venv venv
   ```

   Výsledkom príkazu bude, že v projekte sa vám vytvorí nový priečinok s názvom `venv/`. Okrem toho v pravom dolnom rohu sa zobrazí okno s informáciou, že VS Code zistilo, že sme vytvorili nové virtuálne prostredie a či ho chceme aktivovať. Potvrdíme, že áno.

5. Vypnite terminál a spustite nový. Aktuálne by mal riadok vyzerať v tvare:

   ```
   (venv) $
   ```

   Reťazec `venv` hovorí o tom, že používame virtuálne prostredie.

6. Aktuálne už zostáva len nainštalovať potrebné knižnice. To zabezpečíme spustením príkazu:

   ```
   (venv) $ pip install -r requirements.txt
   ```

7. Úspešnosť inštalácie overíme spustením demo aplikácie. Do príkazového riadku stačí napísať príkaz:

   ```
   (venv) $ python3 main.py
   ```

   Ak ste postupovali správne, v termináli sa vám zobrazí, že aplikácia počúva na porte `8000` (`http://0.0.0.0:8000`). Spustite prehliadač na adrese `http://localhost:8000`. Ak je všetko v poriadku, v prehliadači uvidíte reťazec:

   ```
   "Hello world!"
   ```

Krok 1. Čo budeme potrebovať?

Ešte predtým, ako sa pustíme do tvorby aplikácie, si pripravíme prostredie pre prácu. Konkrétne budeme potrebovať:

- editor kódu jazyka Python (je možné použiť aj Thonny, ale odporúčam IDE, ktoré zvláda prácu s viacerými súbormi a nemá problém s automatickým dopĺňaním kódu),
- stiahnutú kostru projektu z priečinku `app-template/`,
- nainštalované balíčky zo súboru `requirements.txt`.

Krok 2. Hello world!

Úloha 2.1 Otvorte v editore kódu súbor `main.py`.

```
from fastapi import FastAPI
import uvicorn
app = FastAPI(title="Pokédex")
@app.get("/")
def hello():
    return "Hello world!"
if __name__ == "__main__":
    uvicorn.run(
        "main:app",
        # port, na ktorom sa aplikácia spustí,
        # default=8000
        port=8080,
        # bude akceptovať komunikáciu z akejkoľvek IP adresy,
        # default=127.0.0.1
        host="0.0.0.0",
        # v prípade zmeny súboru sa aplikácia automaticky
        # reštartne, default=False
        reload=True,
    )
```

Volanie `uvicorn.run()` zabezpečí spustenie webovej aplikácie s týmito vlastnosťami:

- `port=8080` – aplikácia bude prístupná na porte `8080`; parameter nie je povinný a ak by sme ho neuviedli, aplikácia sa spustí na porte `8000`,
- `host="0.0.0.0"` – aplikácia bude akceptovať spojenia z akejkoľvek IP adresy; parameter nie je povinný a ak by sme ho neuviedli, bude možné k aplikácii pristupovať len z IP adresy `127.0.0.1`,
- `reload=True` – v prípade, že urobíme zmenu v ktoromkoľvek súbore, aplikácia sa sama aktualizuje; parameter nie je povinný a ak by sme ho neuviedli, jeho predvolenou hodnotou bude `False`.

Poznámka: Ak budete pre vývoj používať ľahký editor kódu Thonny, tak **program nevypínajte stlačením červeného tlačidla**! Miesto toho

stlačte v termináli klávesovú skratku `CTRL+C`. Program totiž zostane spustený na pozadí a pri pokuse o jeho opätovné spustenie dostanete chybovú hlášku o tom, že port je už obsadený. To isté sa stane aj vtedy, ak omylom stlačíte tlačidlo spustiť, keď je program už spustený. Rýchlym riešením v rámci workshop-u je číslo portu zmeniť na iný.

Poznámka pre inštruktora: Pretože čas je najväčší nepriateľ tohto workshopu a ak nepoužívate niektoré z veľkých prostredí, ako napr. VS Code alebo PyCharm, do modulu `main.py` môžete vložiť všetky potrebné importy pre celú aplikáciu už teraz. Toto vám ušetrí čas a nebudete musieť zmätočne behať hore a dolu v kóde vždy, keď začnete používať novú funkciu. Miesto toho môžete študentov vždy len upozorniť, z ktorého modulu boli importované. Ak však budete pre vývoj používať profesionálne vývojové prostredie, nie je potrebné vymenovať všetky importy dopredu – tieto prostredia budú v pravý čas vedieť správny modul importovať samé.

Zoznam všetkých potrebných systémových importov sa nachádza v tomto fragmente:

```
import uvicorn
from fastapi import FastAPI, HTTPException, Request
from fastapi.responses import HTMLResponse
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates
from sqlmodel import create_engine, select, Session, or_
from sqladmin import Admin
```

Importovať budete len moduly, ktoré budú vytvorené neskôr.

Úloha 2.2 Spustite a overte správanie vytvorenej aplikácie.

Po spustení bude aplikácia dostupná na adrese `http://0.0.0.0:8080`, resp. na adrese `http://localhost:8080`, ktorú budeme používať aj my. Ak otvoríme prehliadač na tejto adrese, uvidíme v ňom text:

```
Hello world!
```

Lektor: Viac nám však povie pohľad na hlavičku a telo odpovede. Tie si vieme zobraziť napríklad nástrojom `httpie` z príkazového riadku spustením príkazu:

```
$ http http://localhost:8080
```

Výsledkom bude táto HTTP hlavička spolu s jej telom:

```
HTTP/1.1 200 OK
content-length: 14
content-type: application/json
date: Fri, 08 Nov 2024 23:57:35 GMT
server: uvicorn
"Hello world!"
```

Ako je možné vidieť, typ odpovede (kľúč `content-type` v hlavičke) je v tomto prípade JSON dokument (hodnota `application/json`) a nie HTML dokument, ako je tomu napríklad v prípade mikro webového rámca Flask alebo Django. To je dané tým, že rámec FastAPI je primárne určený na tvorbu HTTP REST API.

A práve vytvorením jednoduchého HTTP REST API budeme pokračovať.

Krok 3. Modelujeme Pokémona

Zoznam všetkých Pokémonov je uložený v súbore `pokedex.sqlite`. Databáza obsahuje všetkých *801* aktuálne známych Pokémonov. Na komunikáciu s databázou SQLite budeme používať rámec SQLModel.

Lektor: Bližšie sa na uložených Pokémonov môžeme pozrieť pomocou SQLite klienta `litecli`. Databázu otvoríme príkazom:

```
$ litecli pokedex.sqlite
```

V databáze sa nachádza len jedna tabuľka. To si môžeme overiť príkazom:

```
pokedex.sqlite> .tables
```

Schému tabuľky si zobrazíme príkazom:

```
pokedex.sqlite> .schema pokemon
```

Zoznam všetkých Pokémonov je možné získať nasledovným SQL príkazom:

```
pokedex.sqlite> SELECT * FROM pokemon;
```

Keďže je ale záznamov veľa a každý z nich má veľa atribútov, overíme celkový počet záznamov príkazom:

```
pokedex.sqlite> SELECT COUNT(*) FROM pokemon;
+----------+
| count(*) |
+----------+
| 801      |
+----------+
1 row in set
Time: 0.004s
```

Obr. 3: Pikachu

Úloha 3.1 V koreňovom priečinku aplikácie vytvorte súbor `models.py`.

Do tohto súboru zapíšeme modely, ktoré budeme vytvárať. V našom prípade budeme pracovať len s jedným modelom, ktorým opíšeme Pokémona.

Poznámka: Pre väčšie projekty je však výhodnejšie vytvoriť samostatný balík (package) s názvom `models` a každý model do neho uložiť ako samostatný modul (súbor).

Úloha 3.2 Vytvorte triedu `Pokemon`, ktorá bude potomkom triedy `SQLModel` a bude reprezentovať model Pokémona.

Aby sme rozumeli tomu, čo je to model, môžeme si zobrať na pomoc hraciu/zberateľskú *Pokémon* kartu:

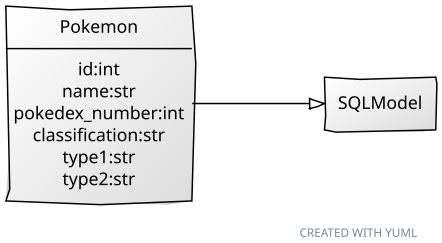
Na karte sa nachádza niekoľko údajov o Pokémonovi Pikachu. Menovať môžeme napr.:

- meno – `Pikachu`,
- číslo Pokémona v Pokédexe – `025`,
- klasifikácia – `Mouse Pokémon`,
- výška – `1.04"`,

- hmotnosť – `13.2lbs`,
- typ pokémona – `electric` (žltá karta, resp. blesk v pravom hornom rohu).

Z týchto vlastností uvedených na karte môžeme vytvoriť vlastný model, ktorý bude opisovať ľubovoľného *Pokémon*-a. Model vytvoríme ako **triedu**, ktorá bude obsahovať len jednotlivé vlastnosti (atribúty) *Pokémon*-a.

Trieda `Pokemon`, ktorá bude reprezentovať náš model bude potomkom triedy `SQLModel` a jej **diagram tried** sa nachádza na nasledujúcom obrázku:



Obr. 4: Model Pokemon

Model, ktorý bude reprezentovať Pokémona, bude mať (zatiaľ) len základné vlastnosti. Dôležité však je, aby ich meno zodpovedalo názvu príslušnej vlastnosti v databáze. Konkrétne sa bude jednať o tieto vlastnosti:

- `id` – primárny kľúč záznamu,
- `name` – meno Pokémona,
- `pokedex_number` – číslo Pokémona v Pokédex-e,
- `classification` – klasifikácia / zaradenie Pokémona,
- `type1` – typ Pokémona,
- `type2` – typ Pokémona.

Následne vytvoríme triedu `Pokemon`, ktorá bude reprezentovať model pre každého Pokémona:

```
from sqlmodel import Field, SQLModel
class Pokemon(SQLModel, table=True):
    id: int | None = Field(default=None, primary_key=True)
    name: str
    pokedex_number: int
    classification: str
    type1: str
    type2: str
```

Poznámka: Názov triedy identifikuje rovno aj názov tabuľky v databáze. To znamená, že každý Pokémon, ktorý bude reprezentovaný pomocou vytvoreného modelu (triedy) `Pokemon` sa bude nachádzať v tabuľke `pokemon`. Ak by sme však potrebovali inú tabuľku, môžeme do modelu pridať atribút `__tablename__` s názvom požadovanej tabuľky, napr.:

```
__tablename__ = "pokemons"
```

Krok 4. Admin rozhranie

Ak chceme pracovať s údajmi uloženými v databáze, musíme sa k databáze pripojiť z našej aplikácie. V tomto kroku preto vytvoríme spojenie s databázou spolu s admin rozhraním z modulu SQLAlchemy Admin.

Úloha 4.1 Vytvorte spojenie s databázou.

Na začiatok súboru `main.py` vytvoríme premennú `engine`, pomocou ktorej sa budeme vedieť spojiť s databázou.

Obsah súboru `main.py` upravíme nasledovne:

```
from sqlmodel import create_engine
app = FastAPI(title="Pokédex")
engine = create_engine('sqlite:///pokedex.sqlite')
```

Úloha 4.2 Vytvorte rozhranie pre administrátora pomocou modulu `sqladmin`.

Admin rozhranie vytvoríme veľmi jednoducho: pod riadok, v ktorom sme vytvorili objek `engine` vytvoríme admin rozhranie riadkom:

```
from sqladmin import Admin
app = FastAPI(title="Pokédex")
engine = create_engine("sqlite:///pokedex.sqlite")
admin = Admin(app, engine)
```

Úloha 4.3 Overte funkčnosť admin rozhrania otvorením adresy `http://localhost:8080/admin/` v prehliadači.

Pokiaľ je všetko v poriadku, v prehliadači sa vám zobrazí prázdna obrazovka admin rozhrania.

Úloha 4.4 V súbore `models.py` vytvorte model pre zobrazenie Pokémonov v admin rozhraní.

V admin rozhraní zatiaľ nič nevidíme. Aby sme v ňom mohli vidieť Pokémonov z databázy, musíme pre ne vytvoriť samostatný model. V súbore s modelmi preto vytvoríme nový model, ktorým opíšeme, čo a ako sa má v admin rozhraní zobrazovať.

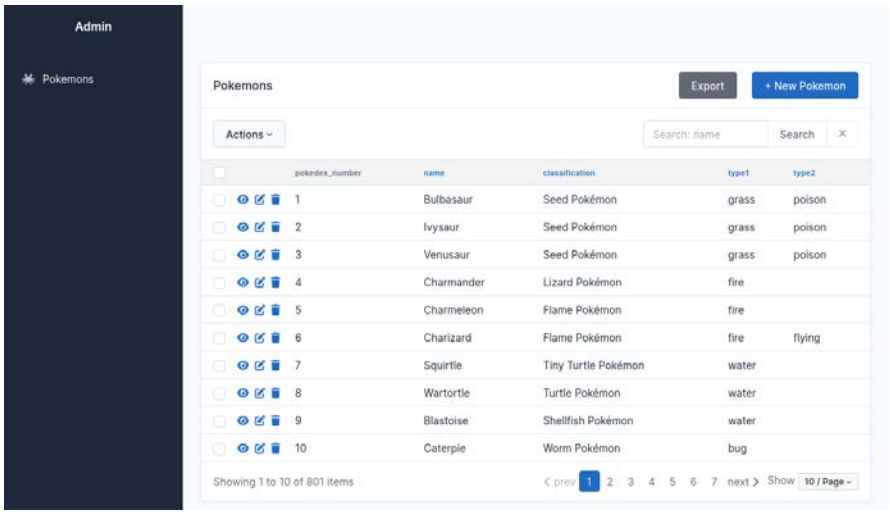
Model nazveme `PokemonAdmin`. Tento model bude potomkom triedy `ModelView` a bude opisovať admin rozhranie modelu `Pokemon`. Kód bude vyzerať napríklad takto:

Obr. 5: Prázdne admin rozhranie

```
from sqladmin import ModelView
class PokemonAdmin(ModelView, model=Pokemon):
    page_size = 20
    icon = "fa-solid fa-spaghetti-monster-flying"
    column_searchable_list = [Pokemon.name]
    column_sortable_list = [Pokemon.name,
        Pokemon.classification, Pokemon.type1, Pokemon.type2]
    column_list = [
        Pokemon.pokedex_number,
        Pokemon.name,
        Pokemon.classification,
        Pokemon.type1,
        Pokemon.type2
    ]
```

Poznámka: Možnosti konfigurácie zobrazenia modelu v admin rozhraní sú široké. O všetkých možnostiach sa dozviete na stránkach dokumentácie modulu.

Úloha 4.5 Pridajte model `PokemonAdmin` do admin rozhrania.



Obr. 6: Admin rozhranie pre Pokémonov

Aby sme v admin rozhraní videli našich Pokémonov z databázy, musíme v ňom pre Pokémonov vytvoriť samostatný pohľad. To urobíme volaním metódy `.add_view()` nad objektom admin rozhrania v súbore `main.py` takto:

```
from models import PokemonAdmin
engine = create_engine("sqlite:///pokedex.sqlite")
admin = Admin(app, engine)
admin.add_view(PokemonAdmin)
```

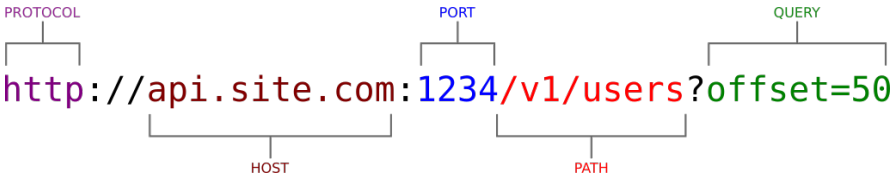
Ak teraz otvoríte prehliadač na adrese `http://localhost:8080/admin/`, uvidíte v bočnom stĺpci názov nášho modelu. Ak naň kliknete, uvidíte zoznam všetkých Pokémonov, ktorých máme v databáze.

Poznámka pre inštruktora: Posledné dve úlohy odporúčam kombinovať. Začať len s premennou `column_list`, ktorá zobrazí príslušné stĺpce o Pokémonovi a následne pridávať a modifikovať ostatné premenné modelu `PokemonAdmin`. A samozrejme bez nutnosti reštartovať aplikáciu – vždy stačí admin rozhranie len obnoviť v prehliadači.

Krok 5. Získanie zoznamu všetkých Pokémonov

V tomto kroku vytvoríme tzv. **endpoint**, pomocou ktorého pošleme klientovi zoznam všetkých Pokémonov. Tento endpoint bude dostupný na adrese `/api/pokemons` a zoznam Pokémonov klientovi pošleme vo formáte JSON.

Endpoint reprezentuje časť URL adresy, ktorú voláme cesta (z angl. path). Keď vytvoríme HTTP požiadavku na takúto adresu, v odpovedi dostaneme zodpovedajúce údaje, ktorými bude v našom prípade zoznam všetkých známych Pokémonov.



Obr. 7: Komponenty URL adresy

Vo výsledku sa samozrejme zavolá funkcia, ktorá sa nazýva **path operation**. A práve funkciu, ktorá vráti zoznam známych Pokémonov, vytvoríme v tomto kroku.

Úloha 5.1 Vytvorte funkciu `get_pokemon_list()`, ktorá vráti (zatiaľ) prázdny zoznam Pokémonov.

Samotná funkcia bude vyzerať takto:

```
def get_pokemon_list():
    return []
```

Aby sme z nej však spravili funkciu typu *path operation*, použijeme nad ňou dekorátor `@app.get()` takto:

```
@app.get("/api/pokemons")
def get_pokemon_list():
    return []
```

O tomto dekorátore platí:

- Jeho parametrom je endpoint (resp. cesta). Táto cesta je potrebná preto, aby rámec *FastAPI* vedel, kedy má túto funkciu spustiť.
- Dekorátor `.get()` reprezentuje metódu HTTP protokolu, ktorou je možné k tejto funkcii pristúpiť. Ak klient použije inú HTTP metódu, táto funkcia sa nespustí.

Úloha 5.2 Overte vytvorený endpoint.

Program aplikácie nie je potrebné zastavovať a znovu spúšťať. Stačí len spustiť HTTP klienta na adrese `http://localhost:8080/api/pokemons`:

```
$ http http://localhost:8080/api/pokemons
```

Výsledkom bude prázdny zoznam.

```
HTTP/1.1 200 OK
content-length: 2
content-type: application/json
date: Sat, 03 Sep 2022 23:02:12 GMT
server: uvicorn
[]
```

Úloha 5.3 Upravte funkciu `get_pokemon_list()` tak, aby miesto prázdneho zoznamu vrátil zoznam všetkých Pokémonov, ktorí sa nachádzajú v databáze.

V jazyku SQL by sme zoznam všetkých Pokémonov získali príkazom

```
SELECT * FROM pokemon;
```

Tento príkaz vieme prepísať do jazyka Python pomocou ORM príkazov, ktoré poskytuje rámec SQLModel. Tým pádom miesto kurzoru, z ktorého musíme výsledky vytiahnuť a ručne z nich spraviť objekty typu `Pokemon` vo výsledku rovno dostanem zoznam týchto objektov.

Zodpovedajúci zápis bude vyzerať takto:

```
from sqlmodel import select
from models import Pokemon
# SELECT * FROM pokemon;
statement = select(Pokemon)
```

Aby sme ho však spojazdnili, potrebujeme otvoriť spojenie s databázou a po vykonaní príkazu ho zasa zatvoriť. Na tento účel vytvoríme objekt typu `Session`. Výsledný pohľad bude vyzerať nasledovne:

```
@app.get("/api/pokemons")
def get_pokemon_list():
    statement = select(Pokemon)
    session = Session(engine)
    pokemons = session.exec(statement).all()
    session.close()
    return pokemons
```

Poznámka: Pracovať s databázou je možné viacerými spôsobmi. To, pre ktorý spôsob sa rozhodnete, závisí od skúseností vašich účastníkov. Pre prácu s objektom typu `Session` môžete použiť napr. tzv. *kontext provider* alebo použiť *dependency injection*. Detaily použitia nájdete v dokumentácii knižnice SQLModel.

Nie je však vždy veľmi praktické vracať zoznam úplne všetkých položiek, ktorých môžu byť v rozličných databázach tisíce až milióny. Zoznam výsledkov preto obmedzíme na *50*. V podstate urobíme niečo, čo by sme v jazyku SQL zapísali ako:

```
SELECT * FROM pokemon LIMIT 50;
```

Výsledná podoba pohľadu bude teda vyzerať takto:

```
@app.get("/api/pokemons")
def get_pokemon_list():
    statement = select(Pokemon).limit(50)
    session = Session(engine)
    pokemons = session.exec(statement).all()
    session.close()
    return pokemons
```

Poznámka: Tento problém sa zvykne riešiť pomocou **stránkovania**. Stránkovanie si je možné vytvoriť ručne alebo je možné použiť existujúci modul FastAPI Pagination.

Krok 6. Získanie detailu o konkrétnom Pokémonovi

Na získanie detailov o konkrétnom Pokémonovi vytvoríme endpoint s parametrom, ktorým bude číslo Pokémona z Pokédexu. Takýto endpoint bude vyzerať takto: `/api/pokemons/{pokedex_number}`. Funkcia, ktorá bude tento endpoint reprezentovať, sa bude volať `get_pokemon_detail()`.

Úloha 6.1 Vytvorte funkciu `get_pokemon_detail()`, ktorá vráti prázdny slovník.

Funkcia bude mať tento parameter:

- `pokedex_number` – identifikačné číslo Pokémona v Pokédexe.

```
@app.get("/api/pokemons/{pokedex_number}")
def get_pokemon_detail(pokedex_number: int):
    return {}
```

Úloha 6.2 Otestujte vytvorený pohľad.

Ak vytvoríme požiadavku s korektnou cestou, v odpovedi dostaneme prázdny JSON dokument. Napríklad:

```
$ http http://localhost:8080/api/pokemons/1
HTTP/1.1 200 OK
content-length: 2
content-type: application/json
date: Sat, 09 Nov 2024 03:56:32 GMT
server: uvicorn
{}
```

Ak však miesto celočíselného identifikátora použijeme iný údajový typ, dostaneme pekné chybové hlásenie. Ak sa napríklad miesto číselného identifikátora budeme pýtať na reťazec `pikatchu`:

```
$ http http://localhost:8080/api/pokemons/pikatchu
```

Výsledkom bude chyba:

```
HTTP/1.1 422 Unprocessable Content
content-length: 164
content-type: application/json
date: Wed, 02 Jul 2025 12:51:51 GMT
server: uvicorn
{
    "detail": [
        {
            "input": "pikatchu",
            "loc": [
                "path",
                "pokedex_number"
            ],
            "msg": "Input should be avalid integer,
                    unable to parse string as an integer",
            "type": "int_parsing"
        }
    ]
}
```

Úloha 6.3 Upravte funkciu `get_pokemon_detail()` tak, aby miesto prázdneho JSON objektu vrátila objekt s dátami o príslušnom Pokémonovi.

V jazyku SQL by sme detaily o Pokémonovi s číslom v Pokédexe *25* získali príkazom

```
SELECT * FROM pokemon WHERE pokedex_number=1;
```

Tento SQL príkaz prepíšeme do jazyka Python pomocou ORM takto:

```
# SELECT * FROM pokemon WHERE pokedex_number=1;;
statement = select(Pokemon).
    where(Pokemon.pokedex_number == pokedex_number)
```

Nakoniec už len aktualizujeme funkciu podobne, ako tomu bolo pri získaní zoznamu Pokémonov:

```
@app.get("/api/pokemons/{pokedex_number}")
def get_pokemon_detail(pokedex_number: int):
```

```
    session = Session(engine)
    statement = select(Pokemon).
      where(Pokemon.pokedex_number == pokedex_number)
    pokemon = session.exec(statement).one()
    session.close()
    return pokemon
```

Úloha 6.4 Overte správnosť vašej implementácie.

Ak napríklad požiadate o detaily Pokémona s identifikačným číslom v Pokédexe *42*:

```
$ http http://localhost:8080/api/pokemons/42
```

dostaneme v odpovedi objekt s informáciami o Pokémonovi s menom *Golbat*:

```
HTTP/1.1 200 OK
content-length: 77
content-type: application/json
date: Sun, 04 Sep 2022 00:56:06 GMT
server: uvicorn
{
    "classification": "Bat Pokémon",
    "id": 42,
    "name": "Golbat",
    "pokedex_number": 42,
    "type1": "poison",
    "type2": "flying"
}
```

Ak sa však opýtame na neexistujúceho Pokémona, napr. s číslom *999*, dostaneme v odpovedi HTTP status kód `500`:

```
HTTP/1.1 500 Internal Server Error
content-length: 21
content-type: text/plain; charset=utf-8
date: Sun, 04 Sep 2022 00:57:42 GMT
server: uvicorn
Internal Server Error
```

čo nie je veľmi sexi. :-(

Úloha 6.5 Zabezpečte, aby v prípade, ak Pokémon s daným číslom neexistuje, mala HTTP odpoveď stavový kód `404`.

Riešiť tento problém môžeme v princípe dvoma spôsobmi:

1. zachytením vzniknutej výnimky `sqlalchemy.exc.NoResultFound`, alebo,

2. miesto metódy `.one()`, ktorá v prípade, že výsledkom dopytu nie je práve jeden objekt, skončí s výnimkou, použijeme metódu `.one_or_none()`, ktorá vráti hodnotu `None`.

V prípade druhého spôsobu bude upravený pohľad vyzerať takto:

```
from fastapi import HTTPException

@app.get("/api/pokemons/{pokedex_number}")
def get_pokemon_detail(pokedex_number: int):
    session = Session(engine)
    statement = select(Pokemon).
        where(Pokemon.pokedex_number == pokedex_number)
    pokemon = session.exec(statement).one_or_none()
    session.close()
    if pokemon is None:
        raise HTTPException(status_code=404,
          detail="Pokemon not found.")
    return pokemon
```

Krok 7. Domovská stránka v HTML pomocou šablónovacieho systému Jinja

Pri prístupe na adresu http://localhost:8080, ktorá reprezentuje domovskú stránku Pokédexu, dostaneme odpoveď typu JSON. V tomto kroku preto vytvoríme domovskú stránku Pokédexu vo formáte HTML. Pre jej reprezentáciu použijeme šablónovací systém Jinja.

Obr. 8: Jinja Logo

Úloha 7.1 Zmeňte typ dokumentu odpovede pre cestu `/` na HTML.

Predvolene je typ výsledného dokumentu JSON. Ak chceme zmeniť typ dokumentu odpovede na HTML, do dekorátora funkcie ošetrujúcej cestu pridáme parameter `response_class=HTMLResponse`.

```
from fastapi.responses import HTMLResponse
@app.get("/", response_class=HTMLResponse)
```

```
def hello():
    return "Hello world!"
```

Typ dokumentu uvidíme v hlavičke odpovede:

```
HTTP/1.1 200 OK
content-length: 12
content-type: text/html; charset=utf-8
date: Sun, 04 Sep 2022 02:13:25 GMT
server: uvicorn
Hello world!
```

Úloha 7.2 Inicializujte šablónovací systém Jinja pre použitie v aplikácii. Potrebujeme inicializovať dve veci:

1. šablónovaciemu systému Jinja potrebujeme povedať, v ktorom priečinku sa nachádzajú šablóny stránok, a,
2. povedať rámcu FastAPI, v ktorom priečinku sa budú nachádzať statické súbory (ako napr. obrázky, CSS štýly, JavaScript-y, a pod.).

```
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates
from fastapi.staticfiles import StaticFiles
app = FastAPI(title="Pokédex")
app.mount("/static", StaticFiles(directory="static"),
    name="static")
jinja = Jinja2Templates(directory="templates")
```

Úloha 7.3 Nahraďte funkciu `hello()` funkciou `homepage()`, ktorá pri prístupe klienta na domovskú stránku Pokédexu vráti v odpovedi HTML stránku zo šablóny `home.tpl.html`.

Funkcii `homepage()` pridáme parameter `request`, ktorý je typu `Request`. Ten je potrebný pri skladaní šablóny šablónovacím systémom Jinja.

Funkcia bude tentokrát vracať objekt šablóny, ktorá má dva parametre:

1. názov súboru šablóny, ktorá má byť vrátená, a,
2. slovník `context` obsahujúci údaje, ktoré sa majú v šablóne zobraziť; v našom prípade sa jedná o kľúče
    1. `request` – objekt požiadavky `request`, a,
    2. `title` – názov webovej stránky.

Výsledná funkcia `homepage()` bude vyzerať nasledovne:

```
from fastapi import Request
from fastapi.responses import HTMLResponse
@app.get("/", response_class=HTMLResponse)
def homepage(request: Request):
    context={
        'request': request,
        'title': 'Vitajte | Pokédex'
    }
    return jinja.TemplateResponse('home.tpl.html', context)
```

Úloha 7.4 Otestujte vytvorenú implementáciu.

Ak ste postupovali správne, po otvorení adresy `http://localhost:8080/` sa vo webovom prehliadači zobrazí domovská stránka v HTML formáte.

Obr. 9: Pokédex: Domovská stránka

Krok 8. Automagická dokumentácia REST API

Pre vytvorené REST API rámec FastAPI automaticky generuje dokumentáciu v dvoch formátoch:

1. Redoc – dostupný na adrese `http://localhost:8080/redoc`,
2. OpenAPI/Swagger – dostupný na `http://localhost:8080/docs`.

K odkazom na oba typy dokumentácie sa dostanete cez menu na domovskej stránke.

Pre viac možností, ako je napríklad dokumentácia jednotlivých endpointov, sa pozrite na oficiálnu dokumentáciu rámca FastAPI.

Krok 9. Pokédex ako HTML stránka

Úloha 9.1 Vytvorte funkciu `view_list_of_pokemons()`, pomocou ktorej zobrazíte webovú stránku samotného Pokédex-u.

V podstate len zmodifikujeme funkciu `get_pokemon_list()` a zabezpečíme, aby miesto JSON dokumentu bola vrátená šablóna zo súboru `pokemon-list.tpl.html`. Do objektu kontextu pridáme do kľúča `pokemons` zoznam výsledkov dopytu z databázy.

```
@app.get('/pokedex', response_class=HTMLResponse)
def view_list_of_pokemons(request: Request):
    session = Session(engine)
    statement = select(Pokemon).limit(50)
    pokemons = session.exec(statement).all()
    session.close()
    context = {
        'request': request,
        'title': 'Všetci Pokémoni na jednom mieste | Pokédex',
        'pokemons': pokemons
    }
    return jinja.TemplateResponse('pokemon-list.tpl.html', context)
```

Úloha 9.2 Overte správnosť vašej implementácie.

Ak ste postupovali správne, po otvorení prehliadača na adrese http://localhost:8080/pokedex sa vám zobrazí *50* prvých Pokémonov z Pokédexu.

Obr. 10: Pokémoni v Pokédexe

Krok 10. Detail Pokémona ako HTML stránka

Úloha 10.1 Vytvorte funkciu `view_detail_of_pokemon()`, pomocou ktorej zobrazíte webovú stránku s detailami o konkrétnom Pokémonovi.

Opäť pôjde o modifikáciu funkcie `get_pokemon_detail()`, ktorá vráti informácie o Pokémonovi v podobe HTML dokumentu. Ak sa Pokémon nájde, funkcia vráti šablónu `pokemon-detail.tpl.html`. V opačnom prípade vráti šablónu `404.tpl.html`.

```
@app.get("/pokedex/{pokedex_number}", response_class=HTMLResponse)
def view_detail_of_pokemon(request: Request, pokedex_number: int):
    session = Session(engine)
    statement = select(Pokemon).
      where(Pokemon.pokedex_number == pokedex_number)
    pokemon = session.exec(statement).one_or_none()
    session.close()
    if pokemon is None:
        context = {
            "request": request,
            "title": "Pokémon sa nenašiel | Pokédex"
        }
        return jinja.TemplateResponse("404.tpl.html", context)
    context = {
        "request": request,
        "title": f"{pokemon.name} | Pokédex",
        "pokemon": pokemon,
    }
    return jinja.TemplateResponse("pokemon-detail.tpl.html",
      context)
```

Úloha 10.2 Overte vytvorenú funkciu.

Ak otvoríte prehliadač na adrese http://localhost:8080/pokedex/25, zobrazia sa vám informácie o Pokémonovi *Pikatchu*. Rovnako sa vám detail o Pokémonovi zobrazí po kliknutí na ktoréhokoľvek Pokémona v zozname všetkých Pokémonov.

Upozornenie: Nezobrazia sa však všetky položky, ktoré šablóna vyžaduje. Ak chceme, aby sa zobrazili, potrebujeme rozšíriť model *Pokémona* o chýbajúce položky.

Krok 11. Vyhľadávač Pokémonov

Úloha 11.1 Rozšírte funkciu `view_list_of_pokemons()` o tzv. *query parameter* `query`, pomocou ktorého zobrazíte webovú stránku so zoznamom Pokémonov, ktorých meno obsahuje reťazec tohto parametra.

Upravená funkcia bude vyzerať nasledovne:

```
from sqlmodel import or_
@app.get('/pokedex', response_class=HTMLResponse)
def view_list_of_pokemons(request:Request,query:str|None=None):
    if query is None:
        statement = select(Pokemon).limit(40)
    else:
        statement = (
            select(Pokemon)
            .where(or_(Pokemon.name.ilike(f'%{query}%'),
              Pokemon.id == query))
            .limit(40)
        )
    session = Session(engine)
    pokemons = session.exec(statement).all()
    session.close()
    context = {
        'request': request,
        'title': 'Výsledky hľadania | Pokédex',
        'pokemons': pokemons
    }
    return jinja.TemplateResponse('pokemon-list.tpl.html',
      context)
```

Úloha 11.2 Overte správnosť svojej implementácie.

Ak ste postupovali správne, tak ak do vyhľadávača zadáte napr. `charm`, vo výsledku dostanete dvoch Pokémonov: *Charmandera* a *Charmeleona*.

Ďalšie zdroje

- Pokédex – online pomôcka každého správneho lovca Pokémonov.
- FastAPI – mikro webový rámec jazyka Python.
- SQLModel – knižnica pre interakciu s SQL databázami s objektami v jazyku Python.
- SQLAdmin – admin rozhranie pre modely napísané pomocou knižnice *SQLModel*.

- Jinja – šablónovací systém.
- Kaggle: The Complete Pokemon Dataset – dataset obsahujúci zoznam všetkých Pokémonov.
- Real Python: Build a URL Shortener With FastAPI and Python – návod, ako vytvoriť skracovať adries v štýle bit.ly pomocou mikro webového rámca FastAPI.
- Real Python: Primer on Jinja Templating – návod, ako pracovať a používať šablónovací systém Jinja (v mikro webovom rámci Flask).
- Namakané webináre: Tvorba dynamických webových stránok v jazyku Python, časť 1, časť 2 a časť 3.

Licencia

KONFERENCIA OSSCONF 2024 (SPRÁVA)
REPORT: OSSCONF 2024

Patrik Sleziak

E-mail: patrik.sleziak@gmail.com

V dňoch 2. 7. – 4. 7. 2024 sa na Fakulte riadenia a informatiky Žilinskej univerzity uskutočnil 12. ročník konferencie OSSConf. Konferencia je primárne venovaná otvorenému softvéru vo vzdelávaní, výskume a IT riešeniach. Aktuálny 12. ročník zastrešoval šesť tematických blokov: LaTeX, R a ich priatelia, Open AI, OSS vo vzdelávaní, Open hardvér, Vývoj OSS, Open GIS & Open Data. Nás samozrejme najviac zaujímala posledná menovaná, t. j. sekcia venujúca sa GIS-om, ktorá prebehla 4. 7. 2024.

Po úvodných slovách Mila Ofúkaného (Ministerstvo vnútra SR, Geokomunita) sa s prezentáciou s názvom „Zásuvný modul Dyna Crop“ predstavil prvý rečník Jan Růžička (Cybele, GISMentors). V úvode spomenul aj ďalších dvoch spoluautorov modulu Dyna Crop (t. j. pluginu pre QGIS), ktorými sú jeho žena Kateřina Růžičková, ktorá pôsobí na Katedre geoinformatiky VŠB-TUO a Oto Kaláb z Ostravskej univerzity.

Rečníkove prvé slová patrili skupine GISMentors, http://gismentors.cz/, ktorej jednou z náplní práce je aj vývoj úloh pre QGIS. GISMentors predstavuje nezávislé združenie školiteľov v oblasti GIS a open source riešení. Lektori ponúkajú širokú paletu kurzov, ako napr. úvod do GIS, QGIS, PostGIS, GeoServer, GeoPython a pod. Posledný menovaný kurz som absolvoval aj ja a musím povedať, že to stálo za to. Určite odporúčam. Materiály zo školení sú voľne dostupné, https://training.gismentors.eu/, dajú sa stiahnuť, používať, rozširovať, zároveň je možné sa zapojiť do ich vytvárania a použiť ich pre komerčné aj nekomerčné účely.

Rečník naznačil, že ich vlajkovou loďou je nástroj GISQuick, https://gisquick.org/, ktorý umožňuje publikovať mapový projekt z prostredia QGIS na web. Pár slov bolo adresovaných zásuvnému modulu pre QGIS s názvom GeoData CZ/SK, https://plugins.qgis.org/plugins/czech_slovak_freegeodata. Ide o plugin, ktorý umožňuje pomerne rýchlou cestou sa dostať k rôznym českým/slovenským dátam (napr. chránené územia, komunikácie, sídla, vegetácia,... jednou z novších vecí sú mapy.cz a tieňovaný reliéf z DEMu 5G) ku ktorým by sme sa obyčajne dostali tak, že by sme museli ísť napr. na stránky Ministerstva, do Geoportálu, skopírovať adresu, následne pridať do QGISu, čo by bolo pracné. Ďalší plugin pre QGIS, ktorý rečník spomenul je napr. radiačný toolbox plugin, ktorý vyvíjajú pre Inštitút radiačnej ochrany, ktorý sa v ČR venuje ochrane v oblasti radiačného riadenia alebo plugin NDOP Downloader (autor Oto Kaláb,

https://opengeolabs.github.io/qgis-ndop-downloader/), ktorý umožňuje stiahnutie a import dát z databázy ohrozených druhov (NDOP) ČR.

Postupne rečník prešiel k zásuvnému modulu Dyna Crop, https://plugins.qgis.org/plugins/qgis-world-from-space-plugin-v_0_1_10_beta, ktorému patril názov prezentácie. Modul (t. j. plugin v QGISe) slúži pre komunikáciu s API spoločnosti World From Space. Pracuje s dátami Sentinel s rozlíšením 10 metrov na pixel, sekundárne číta dáta z PlanetScope (3 metre na pixel). Plugin vznikol z toho dôvodu, aby sa tieto dáta čo najrýchlejšie dostali ku klientom, keďže riešenie napr. cez klientskú webovú aplikáciu, ktorá má nejaké limity obmedzenia sťahovať dáta a zobrazovať v QGISe by nemuselo byť vyhovujúce. Pomocou modulu Dyna Crop je možné riešiť úlohy spojené s vlhkosťou pôdy, vegetačnými indexami, obsahom soli v pôde a pod. Modul je postavený na takom princípe, že užívateľ špecifikujete nejaký polygón (záujmové územie/pole), špecifikuje časový rozsah, ktorý chce spracovať, ďalej volí aký index chce spracovať a čo chce vizualizovať (napr. nejaký index), zároveň si vie zvoliť zonáciu, teda priestor rozdeliť na nejaké zóny (napr. časť poľa ohrozená suchom). Keď prebehne spracovanie, dáta sa načítajú a zobrazia v QGISe, buď ako graf alebo raster. Proces výpočtu môže trvať polhodinu, v niektorých prípadoch aj hodinu, záleží aké veľké je záujmové územie a pod. Každopádne, plugin Dyna Crop sa javí ako ako šikovný nástroj, ktorý stojí za vyskúšanie. Dokumentácia je dostupná na https://docs.dynacrop.space/docs/.

V druhom príspevku Róbert Sásik priblížil výučbu open source databázových systémov v geografických informačných systémoch pre študentov odboru geodézia a kartografia na Katedre geodézie Žilinskej univerzity v Žiline. Zdôraznil význam priestorových databáz, databázového systému PostreSql a jeho rozšírenia PostGIS.

Ako tretí v poradí sa predstavil Martin Tuchyňa (MŽP SR), ktorý sa v rámci svojho príspevku s nami podelil o skúsenosti s využívaním softvérových nástrojov s otvoreným zdrojovým kódom v oblasti verejnej správy. Spomenul, že ich práca s open source nástrojmi vychádza z dvoch strán, po prvé sú to požiadavky so strany zákonov, smerníc a právnych aktov, ktoré sú niekedy striktné a tvrdé, po druhé sú to potreby klientov zo strany verejnej správy. Podotkol, že v súčasnosti z pohľadu koordinácie INSPIRE je na Slovensku väčšina bežiacich/vyvíjaných riešení postavená nad open source. Počas prezentácie sa zmienil o prínosoch (napr. úspora verejných zdrojov, spolupráca s externými expertmi) ale aj výzvach (napr. obmedzená komunitná podpora niektorých open source produktov) z hľadiska open source technológií.

Nasledoval rečník Jakub Šperka (STU Bratislava), ktorý sa vo svojej prezentácii zameral na priblíženie otvorených a nízkonákladových IoT riešení vo

vývoji geodetického vybavenia. V úvode naznačil, že bežne v teréne je behanie medzi orientáciami a bodmi vytyčovacej siete a nastavovaním hranolov niekedy časovo náročné. Za cieľ si teda zobral navrhnúť produkt, ktorý by bol schopný automatického natočenia geodetického hranola v rámci vytyčovacej siete požadovaným smerom, čo sa mu aj podarilo.

Ďalším v poradí bol Michal Páleník (Fakulta managementu UK v Bratislave), ktorý predstavil svoj vytvorený LAU1 dataset o nezamestnanosti a demografii krajín Vyšehradskej skupiny dostupný na webstránke Inštitútu zamestnanosti, https://www.iz.sk/en. Užívatelia si dataset vedia priamo zo stránky stiahnuť a použiť pre riešenie rôznych úloh. Autor prizvukoval, že ho potom netreba zabudnúť citovať. Je tam viacero formátov s ktorými sa dá pracovať, napr. CSV (napr. vývoj počtu obyvateľov), SQL (postgisový export), TopoJSON, GeoJSON, JSON, Shapefile a pod. Ako ukážku v prezentácii na spracovanie datasetu použil kombináciu open source programov PostgreSQL, PostGIS a R.

Ako posledný vystúpil Michal Plánka (Vars Brno), ktorý v prezentácii zaujímavým spôsobom formou hry demonštroval potenciál využitia otvorených geografických dát v Česku a open source riešení pre lepšie porozumenie okoliu a efektívnejšie plánovanie a riadenie svojich činností. V rámci prezentácie sme si prešli tri ukážkové príklady: (a) Vytvorenie mapy s animáciou, ktorá vizualizovala index kriminality v určitej časovej rade na podrobnej priestorovej úrovni, (b) Vytvorenie grafiky pre turistické informačné tabule, ktoré budú umiestnené vo vybraných lokalitách na území CHKO Beskydy, (c) Vhodný výber lokality s nejakými špecifickými požiadavkami v Česku.

Na záver, po skupinovej foto rečníkov ešte Milo Ofúkaný spomenul pár blížiacich sa podujatí v roku 2025: 30-ty ročník konferencie GIS Ostrava a 2-hý ročník Geovision (ten sa naposledy konal v roku 2005), ktoré prebehnú na pôde VŠB-TUO.

Posledné slová patria už len organizátorom, ktorým sa opäť konferencia vydarila. Budeme sa tešiť na ďalšie ročníky.

Skupinové foto rečníkov

OHLÉDNUTÍ ZPĚT ZA OSSCONF 2025

REPORT: OSSCONF 2025

Pavel Stříž

E-mail: pavel@striz.cz

Motto: Vracím se myšlenkama do dětství, k úlohám, které byly tehdy mnou jednoznačně neřešitelné...

1. Ohlédnutí ještě dál do minulosti

Ročník 2025 byl pro mě výjimečný. Především proto, že jsem po mnoha a mnoha letech prezentoval výsledky problému, se kterým jsem si myslel, že nepohnu. Vždy jsem si přál, abych pohnul a mohl výsledky v Žilině představit. Některé úlohy to tak mají: dají vám za uši, a nemusí to být ani Riemannova domněnka, už se tuší, že je to hypotéza, či odpověď na otázku, je-li $\mathcal{P} = \mathcal{NP}$. Byl to tento ročník, kdy se to zalomilo! Více viz článek o mracích slov, zde nebudu detaily prozrazovat. Naznačím transformaci jednou ukázkou. Jen nahlas vyřknu, že kdybych se z pracovních, rodinných či osobních důvodů nemohl dalšího ročníku/ročníků zúčastnit, to důležité, subjektivně řečeno, jsem představil.

Poprvé jsem se objevil v Žilině v roce 2010. V roce 2009 jsem pomáhal se sazbou knihy u bratra/nakladatele, a jeho zákazník si všiml, že umím TEXovat a užívám Linux apod., a tak se mě zeptal, jestli nechci přijet do Žiliny, že i oni tam vyrazí. Proč ne, že? A tak to celé u mne začalo.

Cestoval jsem tehdy s Jiřím Rybičkou a Honzou Přichystalem, v uvítacím výboru byl Slavko Fedorik, který se zvládl zastavit i v roce 2025. On si taky nedá říct, musí dojet hlava-nehlava, zdraví-nezdraví! To já jen první vtípek na úvod, neb realita je taková, že Slavko musí na cestování mimořádně opatrně skrz zdraví a museli jsme vytáhnout všechny trumfy, abychom ho do Žiliny dostali.

Poznámka. Na tomto místě by bylo vhodné zmínit, že taková akce je na Slovensku de facto jediná. Našli bychom specializované konference na Python, GIS atd., ale takový mix je ojedinělý. V České republice bychom to mohli přirovnat ke konferenční řadě OpenAlt, viz https://www.openalt.cz, která se odehrává v Brně na FIT VUT, obvykle v listopadu. Ta je však mnohem větší. Podobně na tom nejspíš bude brněnský DevConf. Věřím tomu, že bychom našli něco i v Praze, pravděpodobně LinuxDays, P2D2, OpenSSL, EuroPython a řadu dalších.

Ondřej Ledvinka doporučil příchozí GIS Ostrava, https://gisak.vsb.cz/wordpress/, Applied Geoinformatics, https://isagsymposium.org/, a České uživatelské fórum Copernicus a Dálkový průzkum Země, https://copernicus.chmi.cz/index.php/ceske-uzivatelske-forum-copernicus-a-dalkovy-pruzkum-zeme-2025/.

Tyto akce spadají více či méně do kategorie specializovaných konferencí.

2. O Žilině

Než vám prozradím, co na konferenci zaznělo a co se objevuje v tištěném a rozšířeném online sborníku, viz https://frcatel.fri.uniza.sk/osscconf/zborniky-online, chtěl bych zmínit pár slov o Žilině.

Je na Slovensku ;-), je to úctyhodná metropole budovaná již z doby paleolitu, kterou jsem díky Štefanu Peškovi s chotí a Rudolfu Blaškovi (nabídka bydlení v městské části Vlčince kousek od fakulty před konferencí a na Hlinách V během konference) mohl poznávat několik bonusových dní. Ono se totiž kolegům nelíbilo, že bych měl přespávat v autě, což byl můj původní nápad na konferenční dobrodružství, jak se to docela běžně děje u turistiky v zahraničí.

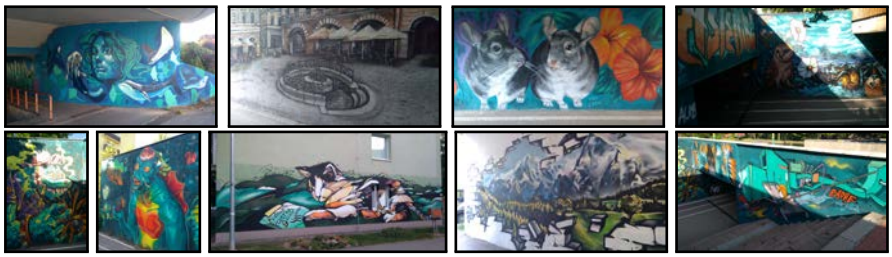


Doprava bez GPS je jen pro místní, ale má francouzská kráska před důchodem Xsara Picasso nakonec vše zvládla. Zde je fotka z návštěvy chaty u Peškových při netradičním sbírání třešní: uříznutí větve a její očesání, obec Višňové

u Žiliny, a z pěšího výletu se Štefanem Peškem na Straník, 769 m.n.m., obec Zástranie u Žiliny.

Musím se pochlubit, že jsem vyřešil svůj letitý problém, a to jak nahodit appku https://www.mapy.cz na starých Androidech, verze 5.0 až 7.0, viz https://mapy-cz.en.uptodown.com/android/versions, kde se bez doinstalování pem souboru, viz tip na https://community.mapy.com/-topic/282/mapy.cz-android-7-nefunkční-připojení/3, a bez zaheslovaného či zapinovaného mobilu nedostanete ke stažení offline map (platit za data na mobilu: to mě ani nehne). Jak to tak bývá, vyřešil jsem to až po konferenci, cestu mi zpříjemnil Android až verze 8.0, kde je vše již automaticky předžvýkané, myšleno nakonfigurované.

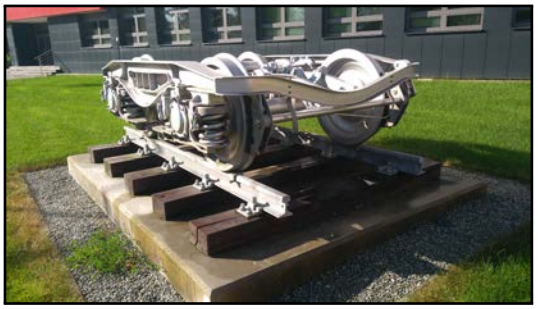
Co mě zaujalo velice, byly graffiti, evidentně od profíků. Místní říkali, že to pak amatéři přešplíchávají. Nu, konkurenční boj je všude, i ve svobodném světě graffitů. Přikládám několik vzorků z podchodů a průchodů paneláky v různých částech města.



Mě ráno pozdravila lama, jak jsem chodil z Hlinů V ze středoškolských internátů různými cestami, viz Farma Žirafa, https://zirafa.sk/farma-zirafa/, kousek od krytého bazénu. Vysokoškolské koleje Veľký Diel se z velké části rekonstruovaly.

Fakulta řízení a informatiky oslaví v září 35 let své existence. Na další straně jsou dvě výstavní fotky před Fakultou strojní, která je jen pár set

metrů od fakulty, kde se akce konává. Musíte jen vědět, kde a jak správně zahnout a neskončit v městském lesoparku Chrasť v opačném směru, pokud se vám výpočetní mobily s GPS zrovna nabíjí. Poučení zní, že sever není vždy po levé ruce. To platí jen tehdy, když jste natočení směrem na východ, nikoliv směrem, kde je zrovna Slunce, třeba když zrovna zapadá. To pak po levé ruce máte opačný jih. „To je chyba v desetinné čárce, pane doktore!“, jak by řekl český komik Zdeněk Izer.


3. O tom letošním ročníku

Lidé se mění, konference se mění. Jsme všichni takoví malí chameleóni či hadi měnící kůži. Nenápadně naznačuji další interní vtípek organizátorů, kteří mi poradili, ať s každou změnou tématu své přednášky, změním design. Tak jsem měnil trička, košile, střih vlasů a vousů, prostě vše, co šlo. Snad to posluchačům pomohlo se zorientovat.

Letos poprvé převzal po Aleši Kozubíkovi žezlo Miroslav Kvaššay a do síně slávy byla poprvé uvedena sekce o Open AI.

Bylo potřeba nahodit nový server a redakční systém WordPress, Peter Sedláček se ujal administrace, neb Miloslav Ofúkaný letos mohl jen částečně pomáhat, plus bylo potřeba upravit web konference výrazně, viz `https://ossconf.fri.uniza.sk`, snad se to podařilo, byť je to hodně neviditelná a nevděčná práce.

Měli jsme obavy, jestli se po kovidu konferenční řada znovu chytne. Snad se ony obavy ukázaly jako liché. Dokonce se podařilo otestovat pár (staro)nových myšlenek, jako přípravu konferenčních triček a propisek. Orgové měli bílá či modrá trička, zvaní přednášející žlutá, další barvy triček bylo možné si zakoupit. Podobně jako sborník v tištěné podobě. Mně krásně sedlo modré 3XL, fotka vpravo.

Loni jsem na konferenci přivezl na rozdávání 3D sudoku kostku a nějaké to elektro, letos jsem nezapomněl a dovezl autorské výtisky mimořádných

čísel České statistické společnosti z let 2020 a 2021, kde se objevuje pár mých článků a příspěvek Rudolfa Blaška o Asymptote. Pokud se na vás výtisk nedostal, stahujte z https://www.statspol.cz/informacni-bulletin/on-line-verze/.

Srdečné pozdravy účastníkům přes poskytnutý rozhovor poslal profesor Vašek Chvátal, významný to matematik a spoluautor programu Concorde TSP Solver, viz https://www.math.uwaterloo.ca/tsp/concorde/, jehož hierarchické algoritmy běží snad úplně všude, kde se problém obchodního cestujícího vyskytuje, např. navštívení lokalit na Google Maps, dráhy 3D tisků, běh CNC strojů ap.

4. Open GIS a Open Data

Úterý, 1. 7. 2025. Dominantou konference bývá Open GIS a Open Data, ta konferenci netradičně začala. Bylo to dané různými časovými omezeními ze stran organizátorů a přednášejících. Akorát Miloslav Ofúkaný nemohl letos dorazit, tak jsme se spolu s Peterem Mártonem snažili mu pomoci. Zde velké díky patří Michalu Páleníkovi, který naznačil směr svého bádání v R již v roce 2024, viz https://www.iz.sk/cr-sr, a nebál se chopit zvané přednášky. V návaznosti na jím užitá data i Michalu Kaukičovi, který testoval vydolovaný poklad, nástroj Marimo, a nezávisle si s daty v Pythonu pohrál, viz https://unemp.feelmath.eu/. Bylo příjemné vidět svět R a Pythonu vedle sebe.

Není v mých silách a prostorových možnostech, abych představil všechny přednášky GIS sekce, ale poněvadž formálně pod TEX sekci spadlo i zpřízněné výpočetní prostředí R, tak naše srdíčka zaplesala, neb v této sekci bylo možné dále vyslechnout přednášku Ondřeje Ledvinky z Prahy o R balíčcích v projektu PERUN, https://www.perun-klima.cz/. A také Ondřeje Vencálka z Olomouce o přípravě mapy konferenční řady ROBUST (robustní statistika), https://www.statspol.cz/konference/robust/, tentokrát R ve spolupráci s knihovnou plotly, https://plotly.com/.

Já jsem představil mapu ČR a SR s erby a věci kolem toho. Plus jsem představil aktuálně řešenou problematiku, a to jak sázet šestiúhelníková loga výpočetního prostředí R do předvoleného mustru. Mám to vyřešené přes Python, přes R to ještě řeším, jak tam vůči roku 2018 toho už moc nejede, viz https://mitchelloharawild.com/blog/user-2018-feature-wall/. Zde je jistá ochutnávka rozkresu.

Mé podklady lze nalézt na https://archive.org/details/@malipivo.

Den uzavřel Missing Maps Mapathon s Peterem Mártonem, viz fotka, a venkovní posezení za fakultou u tradičního guláše. Díky patří především kulinářskému umění Tomáše Majera a dalším, kteří mu byli k ruce.

Více za GIS sekci prosím hledejte ve zprávách od Patrika Sleziaka na https://gisportal.cz/: letošní, https://gisportal.cz/konferencia-ossconf-2025-report/, či loňskou, https://gisportal.cz/konferencia-ossconf-2024-report/. Přetisky naleznete v bulletinku.

5. Open Hardware, Vývoj OSS

Středa 2. 7. 2025. Druhý konferenční den zahájila zvaná přednáška praktika Richarda Faby z Popradu z firmy https://www.famme.sk/ na téma OpenPLC a zvládl představit i „vytúněný“ a programovatelný editor Eshell. O správci souborů a archívů Far Manager následně pohovořil místní Rudolf Blaško. Neuvěřitelnou práci na konferenci odvedl Miroslav Biňas z Košic, když postupně představil tři velké projekty, v této sekci kiosk přes Raspberry Pi. V dalších dnech pak přípravu LaTeXové šablony na univerzitě a později ještě jeden, viz dále.

6. LaTeX, R a jejich přátelé

Mluvě o TeXu. Odpoledne se otevřela TeXová sekce a typografie. Tu zahájil náš aktivní vývojář Michal Hoftich z Prahy. Na můj vtípek, že ho uvádím jako našeho vývojáře číslo 1, s úsměvem odpověděl, že ho předběhl Vítek Starý Novotný, současný editor Zpravodaje CSTUGu, https://www.cstug.cz/bulletin/. Snad ho budeme moci potkat osobně na příštím ročníku. Vedle zmíněné šablony Mirka Biňase, viz https://git.kpi.fei.tuke.sk/tuke/thesis.in.latex, zazněly příspěvky Aleše Kozubíka (PythonTeX), Jiřího Rybičky a kolegů na pokračující výzkum dokumentové gramotnosti, Rudolfa Blaška (Pokročilý TikZ) a došlo i na pár mých maličkostí, byť já už prakticky neTeXuji. To je interní vtípek Petra Olšáka, který o sobě vždy tvrdí, že už

také neTEXuje, viz https://petr.olsak.net/. Možná tvrdí po letech něco jiného, to bohužel nevím, dlouho jsem ho nepotkal.

Má mise, všechny zkonvertovat na LuaLaTeX, snad časem přinese své ovoce. Horší jen bude, jestli budu častěji užívat ConTeXt, to by ten LaTeXový svět i s jeho balíčky šel vážně do kytek. Opět naznačím vyřčený vtípek, že TEXový svět je jak politické švitoření na úrovni parlamentu či senátu, je dobré se držet zdravého středu, ne ultrapravicového Petra Olšáka (autor OpTeXu) ani ne ultralevicového Hanse Hagena (autor ConTeXtu). Ve skutečnosti to tak extrémní v TEXovém světě není, ale dobře se to pamatuje.

Možná bych mohl kolegům nadhodit myšlenku, že se dá koupit jakýsi Microsoft Word ve výborné aktualizované verzi číslo...

V sekci zazněla řada zpřátelených příspěvků. Za optimalizaci byl Milanem Strakou z firmy INO-HUB Energy představen model lineárního programování, jak vyrobit bateriová uložiště ziskově. Plus zazněla přednáška místních kantorů (Marek Ďurana, Miroslav Kvaššay) na vznikající výpočetní podhoubí na žilinské univerzitě. Se Štefanem Peškou jsme si přihřáli kašičku a pohovořili o sudoku problému, který vypadal původně na cca 900 výpočetních let, ale smrskl se až na cca 40 výpočetní let (optimalizace kódu, užití fint a tipů od kolegů), které se po užití cca 340 jader smrskl na mez řešitelům již přijatelnou a počitatelnou (7 týdnů). Vlastně můžeme říci, že to byla jen taková výpočetní jednohubka ve srovnání se současnými požadavky světa AI a bioinformatiky.

Musím upozornit ještě na jednu záležitost. Jedná se o článek ve sborníku Štefana Peška a Zuzany Borčinové o variantě problému obchodního cestujícího vztaženého na drony a jejich rozvoz. Explicitně na to upozorňuji, neb autoři dali přednost jen verzi článkové, nikoliv o tom v této fázi vývoje pohovořit veřejně.

Den uzavřelo posezení v nedaleké pizzérii Kazačok, viz https://www.kazacokpub.sk/.

7. Open AI, OSS ve vzdělávání

Čtvrtek 3. 7. 2025. Poslední den byl taktéž bohatý. Otevřely se sekce OSS ve vzdělávání a nová sekce Open AI. O zkušenostech s testy na Moodlu a sazbou matematiky přes TEX a MathJax pohovořil Rudolf Blaško. Zvané přednášky přednesly především dámy: Monika Spótová z Bratislavy s Markem Klimou o BBC micro:bit v DIGI CAMPu, Eliška Cézová z Prahy o svých dobrých i horších zkušenostech s ChatGPT. Měl jsem možnost přednášku vyslechnout na STAKANu, kolegyně zapracovala připomínky a v Žilině prezentovala ještě bohatší a pestřejší verzi. Je to šikovná kantorka. A v neposlední řadě kolegyně Miroslava Biňase Zuzana Tkáčová z Košic, první vítězka v ka-

tegorii Učitelka Slovenska. To byla naše tzv. V.I.P. Hovořila o konceptech Open AI v dlouhodobém horizontu výuky, především za programování.

Já jsem dost skeptický, na jádro algoritmů a jejich vysvětlení bych stále upřednostňoval člověka znalého věci, nikoliv AI asistenci, která stále vykazuje chybovost. Jsem ta generace prošlá soutěžním programováním s velkou mírou samostudia z knih. Proto jsem si liboval u knihy *Guide to Competitive Programming* Antti Laaksonena s podporou na https://cses.fi/problemset/ a dvoudílné knihy *Competitive Programming 4* od Felixe Halima et al. s odkazy na online podporu uHunt, https://uhunt.onlinejudge.org/, a Kattis, https://open.kattis.com/. Přichází však nová generace, uvidí se.

Nevím, jestli má poznámka, že za nedodání příspěvku budou následovat fyzické tresty (u mužů proutkem přes prsty, u dam přes zadnici), pomohla, ale hezké články zmíněných autorů zvaných přednášek naštěstí dorazily. Je dobře, že články vznikají, je to trvalá památka někde uložená.

Trochu jsem pomáhal s přípravou programu konference. Místo, abych rozhodil přednášky paralelně, tak jsem využil ranních hodin, a tak konference začínala 2. a 3. den už v osm ráno. Nu, jaké je poučení? Ano, nedávat náročné bojové úkoly Pájovi. Ten totiž vede proti nedostatku času nekompromisní partyzánskou válku a využívá všechny možné skulinky. Je dost možné, že je to důvod, proč jsem nikdy na OpenAlt nebyl, neb jsou tam sekce paralelní a mrzelo by mě, že musím něco vynechat.

8. Vítej Kyberbezpečnosti!

Na Mirka Biňase můžeme prozradit, že se ujmul nelehkého úkolu zahájit příští ročník 7. sekci konference, kterou se ctí pojmenoval Kyberbezpečnost. O tom byla jeho třetí přednáška, o bezpečnosti s myšlenkou zkusit ve výuce CTF. Co znamená zkratka CTF nebudu prozrazovat, ale jako výukový koncept v programování či u hardware to zní minimálně napínavě. Tedy ono je to z angličtiny Capture the Flag, ale to neznalému stejně nic neřekne, a pokud řekne, tak nejspíš z her typu Counter Strike 2. Jestli a co se z toho vyklube, to se nechme překvapit na příštím ročníku.

Jeho neúnavná mnoho let trvající práce bude snad takto oceněna (další prací)! Ostatní organizátoři se bezpečnosti štítí, snad nás přesvědčí o užitečnosti a nezbytnosti se bezpečností vážně zabývat. Mně bezpečnost všude zavazí a zdržuje, ale...

9. Workshopy

Velké díky patří Zuzaně Tkáčové s p5.js a Miroslavu Biňasovi (Python servírovaný na mnoho způsobů, viz https://namakanyden.sk/2025/ či

https://github.com/namakanyden) ještě jednou, protože se vedle svých těžkých přednášek chopili náročných workshopů. Tím doplnili druhý a třetí den konference, vedle proběhlého Missing Maps Mapathonu z dne prvního.

Považuji to za významné počiny, neb bylo ukázáno, že lze výuku provádět během konference. V našem případě v místní PC laboratoři, ale bylo by to možné i obecně, neb řada účastníků konferencí má svůj notebook nebo si může v předstihu zajistit. Zde přikládám jednu fotku od Rudolfa Blaška z jejich labáku, maskoti workshopů: had pomáhá tučňáčkovi. Nebo ho škrtí?

Nápad ve vzduchu je, že by z elektronických podkladů workshopů mělo vzniknout mimořádné číslo u statistiků za rok 2025. Bude v tom ještě dost práce za sazbu a nezávislé otestování, ale myslím si, že to bude stát za to!

10. Pokonferenční úlohy

V roce 2024 jsem prezentoval vyřešené úlohy v sudoku, byl jsem šťastný jako blecha, že mám hotovo, a hle, odvezl jsem si dvě nové výzvy: o maximu bloků od Štefana Peška, výsledky byly představeny letos, a ještě o výpočtech tzv. SudoCube od Romana Hajtmanka, problém, který je v mnoha ohledech hraničně těžký a zatím mi nedává spát. O tom snad pohovořím někdy někde jinde, pokud vůbec. Letos to bylo podobné. Po přednášce o mracích slov, kdy mi spadl kámen ze srdce, že je to za mnou, mne jedna z příchozích nadhodila pár svých nápadů s podobným stupněm náročnosti.

11. Myšlenky závěrečné

Pokud čtete tuto zprávu, ale neměli jste možnost se na OSSConf ukázat, asi vám náš svět svobodného a otevřeného softwaru může připadat zvláštní. Pro řadu z nás je to životní filozofie, pro řadu vývojářů je to motor, že lze mít něco krásného, ale netřeba za to platit. A že programování je i umění, o tom by profesor Donald E. Knuth mohl psát knihy, dodávám s úsměvem.

Co se rýsuje příští ročník? Hlavně akce bude, plánů a vizí by bylo. Bude v Žilině na Fakultě řízení a informatiky, proběhne 1. – 3. 7. 2026, ve středu až pátek, a do síně slávy poprvé oficiálně zavítá sekce Kyberbezpečnosti pod vedením Miroslava Biňase z Košic.

Bu- deme se na vás těšit! A třeba opět dojde i na gulášek a dobroty přímo z výroby ze slavkovské pekárny Herold z Jižní Moravy, ... Na viděnou v roce 2026 na konferenci!
♡