Návrh systémů IoT

2. Pokračování v Pythonu. Technologie pro frontend

Stanislav Vítek
Katedra radioelektroniky
České vysoké učení technické v Praze

Obsah přednášky

1. Funkce a OOP v Pythonu
2. HTML
3. CSS
4. Flask

Funkce a OOP v Pythonu

Funkce

- Funkce je obecně pojmenovaný blok kódu
- Funkce je možné opakovaně volat a parametrizovat
- Podobně jako u větvení a cyklů jsou v Pythonu bloky řešeny konzistentním odsazováním

```
def square(x):
    return x*x

a = square(10)
print(a)
```

Další infornace v Jupyter notebooku k přednášce.

Objektově orientované programování

- V tomto *programovacím paradigmatu* popisujeme realitu tak, jak ji normálně vnímáme
- Základní jednotkou popisu je objekt, který vzniká na základě popisu - třídy
- V řeči programátora představuje třída definici datového typu a objekt hodnotu
- Každý objekt má své **atributy** a **metody**
  - atributy jsou vlastnostmi objektu a/nebo data - vnitřní stav
  - metody umožňují manipulaci s atributy a představují schopnosti objektu komunikovat s vnějším světem - definují **rozhraní**

Třídy a objekty

- Třída je vzor, podle kterého se objekty vytváří. Definuje jejich vlastnosti a schopnosti.
- Objekt, který se vytvoří podle třídy, se nazývá instance.
- Instance mají stejné rozhraní jako třída, podle které se vytváří, ale navzájem se liší svými daty (atributy).
- Komunikace mezi objekty probíhá pomocí předávání zpráv.
- Při vytváření objektu je implicitně volána speciální metoda - **konstruktor**
    - konstruktor buď definujeme sami, nebo se vytvoří automaticky

Třídy a objekty v Pythonu

Definice třídy, atributů, konstruktory a jedné metody

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def introduce(self):
    print("My name is ", self.name)
    print("I am ", self.age, "years old")
```

Vytvoření objektů (instancí třídy) a jejich použití:

```
homer = Person("Homer Simpson", 39)
bart = Person("Bart Simpson", 10)
homer.introduce()
bart.introduce()
```

Zapouzdření

OOP stojí na základních třech pilířích:

- **Zapouzdření**
- Dědičnost
- Polymorfismus

Zapouzdření

- Umožňuje skrýt některé metody a atributy, takže jsou použitelné jen pro třídu zevnitř.
- Zapouzdření tedy donutí programátory používat objekt jen tím správným způsobem.
- Rozhraní třídy rozdělí na veřejně přístupné (public) a její vnitřní strukturu (private).

Privátní a veřejné atributy a metody

```
class _Private:
  def __init__(self, name):
    self.name = name

class NotPrivate:
  def __init__(self, name):
    self.name = name
    # I když je něco privátní, stále je možné to používat
    self.priv = _Private(name)

  def _dispaly(self):  # Private
    print("Hello")

  def display(self):   # Public
    print("Hi")
```

Reference na objekt

- V Pythonu se metoda vždy automaticky volá s referencí na objekt, se kterým byla zavolána.
- Tento objekt se předává jako první (implicitní) parametr metody.
- Silnou konvencí je, že se tento parametr nazývá **self**.
- Když voláme metodu objektu, nemusíme self předávat explicitně. Python to za nás udělá automaticky.
- K čemu referenci potřebujeme? Pro identifikaci konkrétní instance třídy

HTML

- HTML (HyperText Markup Language) se používá k vytvoření struktury webové stránky.
    - Co je hypertext? Je to text, který není určen pro lineární čtení
- HTML se skládá z tagů.
    - Tagy jsou obvykle párové, s daty umístěnými mezi značkami.
    - Obvykle jsou tagy odsazeny pro lepší vizualizaci jejich hierarchie, ale jakékoli odsazení je pouze stylistické.
    - Tagy mohou mít také atributy, což jsou datová pole, někdy povinná a někdy volitelná, která poskytují další informace prohlížeči o tom, jak zobrazit data.

Informace o dokumentu - DOCTYPE

- určuje typ dokumentu, instrukce prohlížeči, jak a co má zobrazit
- nejedná se o HTML tag
- musí být jako první v dokumentu

XHTML

- tagy jsou ukončené
- tagy a atributy malými písmeny
- styly a scripty jsou externí soubory
- stránka je validní XML

HTML5

- sémantika
- canvas
- formuláře s možností kontrol a validací
- email, colorpicker, search form
- všechny funkce nepodporují všechny browsery, nutno zkusit
- kniha Dive Into HTML5

Základní struktura HTML dokumentu

obsah webové stránky

```
<html></html>
```

metadata o stránce, které jsou pro prohlížeč užitečné při zobrazování stránky

```
<head></head>
```

název stránky

```
<title></title>
```

tělo stránky

```
<body></body>
```

Atributy těla HTML dokumentu

barva pozadí, barva určena jménem

```
<body bgcolor="pink">
```

barva textu, barvy určena zkráceným nebo plným RGB zápisem

```
<body text="#000" link="#ABCDEF">
```

využití události elementu pro zákaz označení textu myší nebo klávesnicí

```
<body onDragStart="return false" onSelectStart="return false">
```

Blokové elementy

h1, h2, h3, h4, h5, h6

- header, hlavička - nadpisy, důležité pro indexaci roboty
- měly by jít po úrovních, každá stránka by měla mít jen 1 h1 a postupně jít dolů
- v HTML5 už nemusí být 1 pro celou stránku, ale i pro část stránky (tag article, atd., viz kniha)

div

- division = sekce, blok, kontejner (obdélník), seskupuje elementy
- hlavně kvůli formátování pomocí CSS
- default se za ním zalomí řádek

p

- paragraph = odstavec
- základ formátování pro text

hr

- horizontal rule = horizontální oddělovač
- nepárová značka
- vykreslí na stránce horizontální čáru

br

- break = ruční zalomení řádky
- neměli bychom jich používat víc za sebou, špatně se s tím pak pracuje a je to nepřehledné
- Používat místo toho div, p a mezery řešit v CSS

tabulky table, th, tr, td

```
<table>
  <tr>
    <td>buňka</td><td>buňka</td>
  </tr>
</table>
```

seznamy ul, ol, li, dl, dt, dd

```
<ul>
  <li>1. položka seznamu</li>
  <li>2. položka seznamu</li>
</ul>
```

formuláře form, label, input, textarea

Inline elementy

a

- anchor, párová značka
- uvnitř může být text, obrázek
- hypertextový odkaz
  - href = odkaz kam, URL, adresa absolutní nebo relativní
  - pokud je na začátku #, pak je to odkaz na místo v tom samém dokumentu
- target _blank, _top, _parent, _self (default)
- default je odkaz podtržený a barevný (modrý), rozlišuje se barva navštíveného a nenavštíveného odkazu, lze změnit pomocí CSS

```
<a href="http://google.com" target="_blank">Google</a>
```

img

- image, obrázek

```
<img src="images/fig1.png" width="640" height="480"/>
```

span

- inline kontejner, seskupuje prvky, nereprezentuje nic konkrétního
- jako div, ale řádkový
- zejména kvůli formátování

Další značky

Komentáře

```
<!-- text komentáře -->
```

Math ML

- není podporován Chrome, viz seznam, kdo to podporuje: http://caniuse.com/#feat=mathml

MathJax

- http://www.mathjax.org/
- prakticky etalon, umí toho nejvíc, ale dost pomalý
- umí i TeX a MathML

CSS

Proč CSS?

- oddělení vzhledu stránky od obsahu
- přizpůsobení prezentace pro rozdílná zařízení
  - (displej vs. tisk, velikost zobrazovací plochy...)
- postupná specifikace jednotlivých vlastností, kaskádovost specifikací

Připojení stylů ke stránce

- atribut style o styly definované přímo u konkrétních elementů o nejjednodušší, ale postrádáme oddělení stylů od obsahu (se všemi důsledky)

```
<span style="color:red;">červený text</span>
```

- element style o začlenění stylů přímo do obsahu stránky (na "jednom" místě) o možnost využití direktivy @import

```
<style type="text/css"> @import url("adresa");</style>
```

- element link o připojení externích zdrojů ke stránce (nejen styly, ale také favicon, RSS atd.) o možnost definice atributu media

```
<link rel="stylesheet" type="text/css" href="/URL adresa" media="screen,print" />
```

Základní zápis stylů

```
selektor {
  vlastnost1:hodnota1;
  vlastnost2:hodnota2;
}
```

- selektorů lze uvádět více najednou (oddělují se čárkou)
- stejný selektor lze uvést na více místech

Hlavní selektory

- značka
- třída
- ID elementu

Selektory

- atributy **class** a **id**
- preferovány vlastnosti "přesnějších" selektorů
  - (výjimku tvoří použití direktivy **!important**)
- pseudotřídy: **:hover**, **:link**, **:active**, **:visited**, **:first-child**, **:nth-child(n)**, **:not(selector)**
- pseudoelementy: **::first-letter**, **::first-line**, **::before**, **::after**, **::selection**
- viz http://www.w3schools.com/css/css_pseudo_elements.asp

Barva textu

- vlastnost color
- způsoby určení barvy v rámci stylů:
  - pojmenované barvy - např. black, red, blue, white...
  - zápis hexa kódem - např. #000000, #0000ff
    - při opakování stejných hodnot lze použít zkrácený zápis: #00ff11 = #0f1
  - rgb zápisem - např. rgb(0,0,0), rgb(0,0,255)
  - rgba zápisem - např. rgba(0,0,0,0.8), rgba(0,0,255,0.2)

```
p { color: red; }
```

Formátování textu

text-align

- zarovnání textu
- left | center | right | justify

text-decoration

- "dekorace" textu
- none | underline | overline | line-through

text-transform

- transformace velikosti písmen (lepší, než psát nadpisy rovnou velkými písmeny)
- none | uppercase | lowercase | capitalize

LESS

- Leaner Style Sheet, https://www.geeksforgeeks.org/introduction-to-less/

```
@selector: box; //using variables

.@{selector} {
  font-weight: semi-bold;
  line-weight: 20px;
}
```

```
.box {
 font-weight: semi-bold;
 line-weight: 20px;
}
```

SASS

* Syntactically Awesome Style Sheet, https://www.geeksforgeeks.org/css-preprocessor-sass/

```
a {
  color: white;

  // Nesting
  &:hover {
    text-decoration: none;
  }

  :not(&) {
    text-decoration: underline;
  }
}
```

Bootstrap

- https://getbootstrap.com
- frontend toolkit s předpřipravenými bloky, kaskádovými styly a JS moduly
- využívá SASS
- témata: https://themes.getbootstrap.com
- skvělý nástroj pro frontend i administrační nástroje

Renderování dokumentu

- Server side rendering (SSR): vykreslování aplikace na straně klienta nebo univerzální aplikace do HTML na serveru.
- Client side rendering (CSR): vykreslování aplikace v prohlížeči pomocí JavaScriptu, který upravuje DOM.
- Rehydration: "Zavedení" zobrazení JavaScriptu na klientovi tak, aby znovu používaly strom a data DOM vykresleného HTML na serveru.
- Prerendering: spuštění aplikace na straně klienta v době sestavení, aby se zachytil její počáteční stav jako statický HTML.

Flask

- Mikroframework pro tworbu webových aplikací s moduly pro řešení řady úkolů
- Základní aplikace vypadá následovně:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
  return '<h1>Hello, World!</h1>'
```

- Kód spustí instanci HTTP serveru, který poslouchá na defaultním portu 5000
- Speciální proměnná **name** obsahuje název aktuálního modulu Pythonu a pomáhá s nastavením cest

Zpracování požadavků

- Instance aplikace může být použita ke zpracování příchozích webových požadavků a odesílání odpovědí uživateli.
- **@app.route** je dekorátor, který změní běžnou funkci jazyka Python na funkci **Flask view**, která převede návratovou hodnotu funkce na odpověď HTTP, kterou zobrazí klient HTTP, například webový prohlížeč.
- Argumentem **@app.route()** je hodnota '/', která znamená, že tato funkce bude odpovídat na webové požadavky na adresu URL /, což je hlavní adresa URL.
- Funkce zobrazení **hello()** vrátí jako odpověď HTTP řetězec

```
<h1>Hello, World!</h1>
```

Šablony

- Flask používá nástroj Jinja, https://realpython.com/primer-on-jinja-templating/
- Pomocí této knihovny lze vkládat do renderovaného dokumentu hodnoty proměnných nebo parametrizovat

```
<body>
  <h1>Hello, {{ username }}</h1>
</body>
```

Na místo **username** se do šablony dosadí hodnota, která byla předána při volání funkce **render_template**

```
@app.route('/name/<name>')
def process(name):
    return render_template('name.html', username=name)
```

Šablony

- Stránky a šablony lze různě propojovat a zavádět mezi nimi vztahy
- Následující příklad demonstruje vytvoření stránky **home.html** ze šablony **layout.html**

```
<!-- home.html-->
{% extends 'layout.html' %}

{% block content %}
    <div class="container">
        <h1>{{title}}</h1>
        <p>{{description}}</p>
    </div>
{% endblock %}
```

Šablony

```
<!-- layout.html -->
<!doctype html>
<html>

  <head>
    <title>{{title}}</title>
    <meta charset="utf-8">
    <meta name="description" content={{description}}>
    <link rel="shortcut icon" href="/favicon.ico">
  </head>

  <body>
    {% include 'navigation.html' %}
    {% block content %}{% endblock %}
  </body>

</html>
```

Šablony

- Bloků může být v šabloně i více

```
{% extends 'layout.html' %}

{% block css %}
  <link href="{{ url_for('static', filename='css/home.css') }}" rel="stylesheet">
{% endblock %}

{% block content %}
    <div class="container">
        <h1>{{title}}</h1>
        <p>{{description}}</p>
    </div>
{% endblock %}
```

Logika v šablonách

Kromě dosazování hodnot a případně vyhodnocování výrazů je možné pracovat i s běžnými algoritmickými konstrukty, jako je podmínka nebo cyklus.

```
{% if ..... %}
   ......
{% elif ..... %}
   ......
{% else %}
   ......
{% endif %}
```

```
{% for item in list %}
   ......
{% endfor %}
```

Logika v šablonách

```
@app.route('/')
def home():
    """Landing page."""
    nav = [
        {'name': 'Home', 'url': 'https://example.com/1'},
        {'name': 'About', 'url': 'https://example.com/2'},
        {'name': 'Pics', 'url': 'https://example.com/3'}
    ]
    return render_template(
        'home.html',
        nav=nav,
        title="Jinja Demo Site",
        description="Smarter page templates with Flask & Jinja."
    )
```

Logika v šablonách

```
<header>
   <nav>
    {% for link in nav %}
      <a href="{{ link.url }}">{{ link.name }}</a>
    {% endfor %}
  </nav>
</header>
```

Další mikroframeworky

- Flask zpracovává požadavky synchronně, tj. server čeká na dokončení zpracování požadavku
- Další synchronní frameworky:
  - Pyramid, https://trypyramid.com
  - CherryPy, https://docs.cherrypy.dev/en/latest/
  - BottlePy, https://bottlepy.org/docs/dev/
- Python má podporu asychronního zpracování, které je možné využít
  - Tornado, https://www.tornadoweb.org/en/stable/
  - Sanic, https://sanic.dev/en/