Python akademie - lekce 7 - 28.11.2024

07_01: Úvod do funkcí

Zajímavé odkazy z této lekce:

- Oficialní dokumentace všech built-in funkcí v Pythonu
- Oficiální dokumentace k built-in funkci isinstance()
- isinstance pro začátečníky

Obecně k funkcím v Pythonu

V Pythonu už některé funkce znáš a umíš je používat.

Třeba funkce `print` a `enumerate`. To ale nejsou jedinné funkce, které můžeš používat.

```
print("Praha", "Brno", "Ostrava", sep=", ")
```

```
Praha, Brno, Ostrava
```

```
print(tuple(enumerate(("Praha", "Brno", "Ostrava"))))
```

```
((0, 'Praha'), (1, 'Brno'), (2, 'Ostrava'))
```

Obecné rozdělení funkcí v Pythonu:

1. **Zabudované funkce**, (z angl. *built-in functions*), tedy `str`, `int`, `bool`, aj.,
2. **Uživatelské funkce**, (z angl. *user-defined functions*), klíčové slovo `def`.

Největší rozdíl mezi **zabudovanými** a **uživatelskými funkcemi** je v tom, že *zabudované funkce* stačí **spustit pomocí jejich jména**.

Zatímco *uživatelskou funkci* je nejprve nutné **definovat** (vytvořit) a teprve poté **použít** (spustit).

Zabudované funkce

Tyto funkce jsou velkými pomocníky, protože ti umožní zjednodušit různé procesy.

Navíc můžeš jejich použití **doplnit volitelnými argumenty**.

Volitelný argument je objekt, který můžeš (ale nemusíš) zadávat.

Funkce umí pracovat bez něj, případně má dopředu nachystanou nějakou **počáteční hodnotu**.

```
print("Matous", "Marek", "Lukas")
```

```
Matous Marek Lukas
```

Pokud funkci `print` napíšeš **bez argumentů**, s několika různými hodnotami za sebou, tvůj výstup se seřadí za sebe.

Zobraz si nápovědu pomocí ohlášení `print(help(print))`:

```
help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Všimni si, že **argument** `sep` má přednastavenou defaultní hodnotu – mezeru.

Proto jsou jednotlivé hodnoty řazené s mezerou za sebou.

Tuto hodnotu můžeš přepsat podle svých potřeb. Například vypsat jednotlivé hodnoty **pod sebe** pomocí speciálního znaku `\n`:

```
print("Matous", "Marek", "Lukas", sep="\n")  # volitelný (také nepovinný) argument 'sep'
```

Argumenty můžeš používat téměř u všech **zabudovaných funkcí**.

Proto pokud budeš potřebovat pracovat se **zabudovanými funkcemi**, vždy zkontroluj, jestli neobsahují nějaký nepovinný argument, který ti pomůže.

```
help(enumerate)
```

```
jmena = ("Matous", "Marek", "Lukas")
```

```
tuple(enumerate(jmena, start=3))
```

Uživatelské funkce

Můžeš se dostat do situace, kdy žádná z nabízených *zabudovaných funkcí* nedělá přesně to, co potřebuješ.

V takovém případě potřebuješ vytvořit vlastní funkci, která ti bude umět pomoct.

Tvůj úkol je napsat proces, který sečte **všechny číselné hodnoty** uvnitř sekvence.

```
ciselna_rada = (1, 2, 3, 4)
```

```
print(sum(ciselna_rada))
```

```
10
```

Pomocí zabudované funkce `sum` to není žádný problém.

Co když sekvence obsahuje **neočekávaný datový typ**:

```
ciselna_rada = (1, 2, 3, "dva", 4)
```

```
print(sum(ciselna_rada))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_29340/4140620857.py in <module>
----> 1 print(sum(ciselna_rada))

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
soucet_cisel = 0


for cislo in ciselna_rada:
    if isinstance(cislo, str) and not cislo.isnumeric():
        continue
    soucet_cisel = soucet_cisel + int(cislo)
else:
    print(soucet_cisel)
```

```
10
```

[Oficiální dokumentace k built-in funkci isinstance()]

[isinstance pro začátečníky]

Co když ale dostaneš **pět různých sekvencí**?

Můžeš samozřejmě přepsat zápis pro každou sekvenci zvlášť.

Ale co když těch sekvencí bude **100**, **10 000**?

Právě proto existují **uživatelské funkce**, kterou stačí **jedenkrát definovat** a následně spouštět kolikrát potřebuješ:

```
ciselna_r_1 = (1, 2, 3, "a")
ciselna_r_2 = (1, 2, 3, 4)
ciselna_r_3 = (5, 6, 7, 8)
ciselna_r_4 = (9, 10, 11, 12)


# Zatím neznámá syntaxe
def secti_vsechny_cisla(sekvence):
    soucet_cisel = 0

    for cislo in sekvence:
        if isinstance(cislo, str) and not cislo.isnumeric():
            continue
        soucet_cisel = soucet_cisel + int(cislo)
    else:
        print(soucet_cisel)
```

```
secti_vsechny_cisla(ciselna_r_1)
secti_vsechny_cisla(ciselna_r_2)
secti_vsechny_cisla(ciselna_r_3)
secti_vsechny_cisla(ciselna_r_4)
```

```
6
10
26
42
```

Ukázku výše **nemusíš nyní chápat**.

Je tu hlavně pro ilustraci, jak je důležité mít uživatelské funkce.

Předpis funkcí

```
def jmeno_funkce(parametr_1, parametr_2):  # Předpis funkce a parametry funkce
    """Popis ucelu funkce"""               # dokumentace funkce
    ...                                    # odsazený kód
    return parametr_1 * parametr_2         # VOLITELNÉ: vracené hodnoty
```

Jak tedy *uživatelskou funkci* správně používat?

Z jakých kroků se správné použití skládá?

Nejprve musíš funkci:

1. Jednou **definovat** (*vytvořit*),
2. a potom ji můžeš začít opakovaně **spouštět**.

Pořadí je **důležité**! Takže nemůžeš spouštět takovou uživatelskou funkci, kterou **prvně nedefinuješ**.

```
vysledek = scitej_dve_hodnoty(1, 14)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_29340/2423764154.py in <module>
----> 1 vysledek = scitej_dve_hodnoty(1, 14)

NameError: name 'scitej_dve_hodnoty' is not defined
```

```
def scitej_dve_hodnoty(cislo_1, cislo_2):             # POVINNÉ: Předpis funkce a paramet
    """
    Vraci soucet dvou hodnot uvnitr parametru.
    """                                               # VOLITELNÉ: dokumentace funkce
    return cislo_1 + cislo_2                          # VOLITELNÉ: vracené hodnoty


vysledek = scitej_dve_hodnoty(1, 14)
print(vysledek)
```

```
15
```

```
print(scitej_dve_hodnoty(1, 14))
```

```
15
```

```
def scitej_dve_hodnoty(cislo_1, cislo_2):             # POVINNÉ: Předpis funkce a paramet
    """
    Vraci soucet dvou hodnot uvnitr parametru.
    """
    print(cislo_1 + cislo_2)# VOLITELNÉ: dokumentace funkce
#     return cislo_1 + cislo_2


scitej_dve_hodnoty(1, 14)
```

```
15
```

Pokud si předchozí ukázku spustíš, nic se nestane. Je to kvůli tomu, že funkci **pouze definuješ** a nespouštíš.

V příkladu si můžeš všimnout těchto **charakteristických rysů** pro uživatelskou funkci:

1. `def` je *klíčový výraz* označující předpis (definici) funkce,
2. `scitej_dve_hodnoty` je tvoje označení funkce, díky kterému můžeš funkci později spustit (ideálně má představovat účel funkce),
3. `(cislo_1, cislo_2)` v kulaté závorce jsou umístěné **parametry** funkce. Tedy proměnné, se kterými chceš, aby funkce pracovala.
4. `:` předpisový řádek musí být ukončený dvojtečkou (jako u podmínkových zápisů, cyklů, aj.),
5. `"""Vraci soucet dvou .."""` na odsazeném řádku následuje *docstring*, tedy bližší popis účelu funkce (zejména pokud jméno nedostačuje),
6. `return` ohlášení z funkce vrací žádané hodnoty (nemusí být součástí funkce vždy).

Spuštění funkcí

Takže pokud máš funkci definovanou, můžeš ji spouštět kolikrát chceš a kde chceš (samozřejmě potom, co ji definuješ).

```
def scitej_dve_hodnoty(cislo_1, cislo_2):            # POVINNÉ: Předpis funkce a paramet
    """
    Vraci soucet dvou hodnot uvnitr parametru.
    """
#     print(cislo_1 + cislo_2)# VOLITELNÉ: dokumentace funkce
    return cislo_1 + cislo_2


soucet_1 = scitej_dve_hodnoty(1, 14)  # 1. spuštění funkce
soucet_2 = scitej_dve_hodnoty(2, 8)   # 2. spuštění funkce


print(soucet_1, soucet_2, sep="\n")
```

```
15
10
```

```
soucet_1 + soucet_2
```

```
25
```

Chyby na začátek

```
scitej_dve_hodnoty  # zapoměl jsem závorky
```

```
<function __main__.scitej_dve_hodnoty(cislo_1, cislo_2)>
```

```
scitej_dve_hodnoty()  # chybějící vstupní hodnoty, tzv. argumenty
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_29340/1545938848.py in <module>
----> 1 scitej_dve_hodnoty()  # chybějící vstupní hodnoty, tzv. argumenty

TypeError: scitej_dve_hodnoty() missing 2 required positional arguments: 'cislo_1'
and 'cislo_2'
```

```
scitej_dve_hodnoty(1, 9, 5)  # špatný počet argumentů při spouštění
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_29340/2600331097.py in <module>
----> 1 scitej_dve_hodnoty(1, 9, 5)  # špatný počet argumentů při spouštění

TypeError: scitej_dve_hodnoty() takes 2 positional arguments but 3 were given
```

```
scitej_dve_hodnoty 1, 9  # zapoměl jsem závorky
```

```
  File "C:\Users\Radim Jedlicka\AppData\Local\Temp/ipykernel_29340/1028542131.py",
line 1
    scitej_dve_hodnoty 1, 9  # zapoměl jsem závorky
    ^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

```
vysledek = scitej_dve_hodnoty(2, 3)
```

```
print(vysledek)
```

```
5
```

```
def scitani_dvou_hodnot():
    return 100 + 12
```

```
scitani_dvou_hodnot()
```

```
112
```

```
def odcitej_dve_hodnoty(cislo_3, cislo_4):
    return cislo_3 - cislo_4
```

```
vysledek = odcitej_dve_hodnoty(10, 5)
```

```
print(vysledek)
```

```
5
```

Pár detailů pro spuštění funkcí:

1. Funkci *spouštíš* přes její **jméno a kulaté závorky**,
2. při definování, do kulatých závorek píšeš obecné proměnné, **parametry** funkcí (zajišťují obecné použití),
3. při spouštění, do kulatých závorek musíš zapsat skutečné hodnoty, tedy **argumenty** funkcí,
4. argumenty si funkce sama skládá do parametrů podle několika vzorů,
5. pokud má funkce vracet hodnoty, obsahuje ohlášení `return`,

6. vrácenou hodnotu si musíš schovat do proměnné (`soucet_1`, `soucet_2`). Pokud to neuděláš, o součet **přijdeš**.

Vstupy funkcí, rozdělení

Obecně funkce pracuje se **vstupy**.

Tento pojem souhrnně označuje nejen *parametry*, ale také *argumenty*.

Ty jsou potom do funkce dávkované dle několika vzorů.

Rozdíl mezi nimi je následující:

- **parametry** slouží jako obecné proměnné při definici,
- **argumenty** jsou konkrétní hodnoty, které vkládáš při spouštění.

Prohlédni si ukázku:

```
def vytvor_uziv_jmeno(jmeno, prijmeni):
    jmeno = (jmeno.lower()[0] + prijmeni.lower())
    return jmeno

vytvor_uziv_jmeno('Ivan', 'Kral')
```

```
'ikral'
```

```
def vytvor_uziv_jmeno(jmeno, prijmeni):
    return (jmeno.lower()[0] + prijmeni.lower())


print(vytvor_uziv_jmeno('Radim', 'Jedlicka'))
```

```
rjedlicka
```

Co jsou tedy **parametry** a co **argumenty**?

Řádné funkce uživatele

Psaní uživatelských funkcí má ovšem jistá doporučení.

1. Znovu nevymýšlet kolo

Nejprve zkontroluji *zabudované funkce*, pak tvořím vlastní funkci:

```
cisla = (1, 2, 3)


# TAKHLE NE!
def vypocitej_sumu(cisla):
    suma = 0

    for cislo in cisla:
        suma = suma + cislo
    return suma


# TAKHLE ANO!
suma = sum(cisla)
```

2. Na jménu záleží

Popisuje totiž účel funkce (pokud nelze napsat, zapiš *docstring* funkce):

```
# TAKHLE NE!
def email():
    pass

email()


# TAKHLE ANO!
def posli_zpravu():
    pass

posli_zpravu()
```

3. Rozumné množství parametrů

Ideálně **2-3 parametry** (jsou ovšem výjimky):

```
# TAKHLE NE!
def zobraz_nabidku(title, body, tlacitko, datum):
    pass
```

```
# TAKHLE ANO!
def vytvor_popisek(title, body):
    pass

def vytvor_tlacitko(tlacitko):
    pass

def vytvor_datum():
    pass
```

4. Co je psáno, to je dáno

Funkce by měla provádět **jedinnou věc** (jinak je špatně čitelná, pochopitelná, testovatelná):

```
# TAKHLE NE!
def posli_email_seznamu_klientu(klienti):
    for klient in klienti:
        if klient.je_aktivni:
            email(klient)


# TAKHLE ANO!
def jen_aktivni_klienti(klienti):
    return [klient for klient in klienti if klient.je_aktivni]

def posli_email():
    pass
```

5. Počítá se jen to, co je doma

funkce pracuje pouze **s vlastními parametry** (proměnnými):

```
# TAKHLE NE!
oddelovac = "---"
datum = "01.01.2001"

def vytvor_zpravu(autor, zapis):
    vytvor_hlavicku(datum, oddelovac)
    vytvor_text(autor, zapis)


# TAKHLE ANO!
def vytvor_zpravu(autor, zapis):
    oddelovac = "---"
    vytvor_hlavicku(dnesni_datum(), oddelovac)
    vytvor_text(autor, zapis)

def dnesni_datum():
    pass
```

😍 Interpret Pythonu miluje funkce! 😍 Vytváří oddělená prostředí pro proměnné, se kterými efektivněji pracuje.