Proměnné a řídící sekvence

Co tím myslím?

Co myslím řídící sekvencí.

- podmínky (if → elif → else),
- cykly (for nebo while),
- procedury/funkce (def).

Proměnné a řídící sekvence

Proměnné

Proměnné v Pythonu

- Proměnné v Pythonu se dají pojmenovat v podstatě jakoukoli posloupností znaků (až na výjimky).
- Nemusíte Pythonu říkat, jaký má proměnná datový typ; on si to určí sám.
- Táž proměnná může být v průběhu programu různých typů.
- Hodnota se do proměnné ukládá jednoduchým =.
  Pozor! Tohle = **nemá nic společného se stejným symbolem v matematice**.
  Čte se zprava doleva.
  - např. `number = 3` znamená „do `number` dosaď `3`“ a
  - `number = number + 2` znamená „do `number` dosaď `number + 2`“.

Proměnné – příklady

Příklad s čísly

Program

```
first_number = 4
second_number = 5
print(first_number * second_number)
```

vytiskne 20.

Proměnné – příklady

Příklad se stringy

Program

```
first_word = "kocour"
second_word = "kočka"
print(first_word[3] + second_word[-2])
```

vytiskne `"ok"`.

Proměnné – příklady

Příklad se seznamy

Program

```
inner_list = [4, "blb"]
outer_list = ["ano", inner_list, 5, 6]
print(outer_list)
```

vytiskne `["ano", [4, "blb"], 5, 6]`.

Proměnné a řídící sekvence

Podmínky

Podmínky v Pythonu

- Podmínky se píší ve tvaru

```
if nějaká podmínka:
```

  Pro další možnosti pište `elif` (zkráceno z else if) a nakonec `else`.
- Kód, který se má za dané podmínky vykonat, **musí být odsazen!** Ideálně odsazujte klávesou `Tab`.
- Každá (správně napsaná) podmínka je v Pythonu vyhodnocena buď jako pravda (`True`), nebo lež (`False`).

Tvoření podmínky – vnitřek

Uvnitř podmínky budeme nejčastěji používat operátory

- `in` (doslova „v“ – testuje, jestli to nalevo je uvnitř toho napravo)
  - Např. `("s" in "synek") == True`, ale
  - `(3 in [1, 2, 4, 5]) == False`.
- `==` (testuje, jestli je nalevo to samé, co napravo). **Tohle je ten ekvivalent jednoduchého = v matice.** V Pythonu jednoduché = dosazuje do proměnných!
  - Např. `("sova"[2] == "v") == True`.
- `<,>,<=,>=` (porovnání toho, co je nalevo, s tím, co je napravo). Symboly `<=` a `>=` značí „menší nebo rovno“ a „větší nebo rovno“, resp.
  - Např. `(5 > 3) == True` a
  - `("c" <= "f") == True`.

Tvoření podmínky – vnějšek

Vně podmínek budeme používat (logické) operátory

- `not` (doslova „ne“ – logický opak podmínky)
  - Např. `not (5 > 3) == False` a
  - `not ("x" in "kocour") == True`.
    Místo `not ("x" in "kocour")` lze psát (přirozenějc) `"x" not in "kocour"`.
- `and` (doslova „a“ – musí platit obě podmínky)
  - Např. `(5 > 3 and "s" in "synek") == True`,
  - `(3 <= 4 and "x" in "kocour") == False` a
  - `(1 in [2, 3] and "x" in "kocour") == False`.

Tvoření podmínky – vnějšek

Vně podmínek budeme používat (logické) operátory

- or (doslova „nebo“ – musí platit **alespoň** jedna z podmínek)
  - Např. `(5 > 3 or "s" in "synek") == True`,
  - `(3 <= 4 or "x" in "kocour") == True` a
  - `(1 in [2, 3] or "x" in "kocour" == False`.

Příklad – liché číslo

Program, který určuje, jestli je číslo liché, může vypadat třeba takto.

```
number = 5
if number % 2 == 1:
    print(str(number) + " je liché.")
else:
    print(str(number) + " je sudé.")
```

Proměnné a řídící sekvence

Cykly

`for` cyklus v Pythonu

- `for` cyklus se v Pythonu píše

  `for` prvek `in` seznam/n-tice/slovník`:`

  a kód uvnitř cyklu se odsazuje.

  Pozor! Python prochází seznam a n-tici po prvcích, ale **slovník po klíčích**.
- Proměnná pro cyklus se může jmenovat jakkoliv. Python do ní během cyklu postupně dosazuje všechny prvky seznamu/n-tice, klíče slovníku a po jeho konci ji zapomene.
- Důležitá je funkce `range(n: int)`, která vrací seznam přirozených čísel menších než `n`. Např. `range(5) == [0, 1, 2, 3, 4]`.

Příklad – průchod seznamem

Program, který vytiskne každý prvek seznamu krát dva lze napsat jako

```
random_stuff = [1, "hračka", [2, 3], (4, 5)]
for wtv in random_stuff:
    print(wtv * 2)
```

nebo použitím `range` jako

```
random_stuff = [1, "hračka", [2, 3], (4, 5)]
for index in range(4):
    print(random_stuff[index] * 2)
```

`while` cyklus v Pythonu

- `while` cyklus se v Pythonu píše

```
while podmínka:
```

  a obsah cyklu je odsazený.
- Stavění podmínek ve `while` cyklu je stejné jako v `if`.

Příklad – mocniny dvojky

Program, který vypíše všechny mocniny dvojky menší než dané číslo `limit`, může vypadat třeba takhle.

```
limit = 69 ** 69
power_of_two = 2
while power_of_two < limit:
    print(power_of_two)
    power_of_two = power_of_two * 2
```

Proměnné a řídící sekvence

Funkce

Funkce/procedury v Pythonu

- V Pythonu se funkce píší

  ```
  def jméno funkce(jména parametrů):
  ```

  a obsah funkce je odsazený.
- `def` je z angl. define.
- Pro ukončení funkce a vrácení nějaké hodnoty slouží

  ```
  return hodnota
  ```

  Pozor! Funkce **nemusí vracet nic**. Jakmile provede svůj obsah, skončí sama, i když `return` nikam nenapíšete.

Příklady funkcí

Funkce, co dostane jméno a příjmení a vrátí je spojená dohromady, se dá napsat třeba takhle.

```
def whole_name(name, surname):
    return name + " " + surname
```

Další funkce, co dostane věk a připojí za něj „let“, vypadá

```
def age_to_string(age):
    return str(age) + " let"
```

Příklad – využití funkcí z před. slidu

Řekněme, že máme daný seznam `data` trojic (jméno, příjmení, věk), kde jméno a příjmení jsou stringy a věk je int. Pomocí funkcí z předchozího slidu ho pěkně vytiskneme.

```
for (name, surname, age) in data:
    whole_name = whole_name(name, surname)
    age = age_to_string(age)
    print(whole_name + ", " + age)
```