IB113 Úvod do programování a algoritmizace
Přednáška 2

Základní konstrukce
Zásady čitelnosti kódu

Nikola Beneš

25. září 2017

Co bude dnes?

Základní konstrukce v Pythonu

- zopakování z minula a rozšíření
- vstup a výstup
- typy v Pythonu a převody mezi nimi
- základní operace s čísly a řetězci
- cykly podrobněji
- funkce podrobněji: návratové hodnoty, volání funkcí

Čitelnost kódu

- doporučení pro psaní čitelného kódu
- PEP8

Kde hledat nápovědu k funkcím?

Nápověda přímo v Pythonu

- v PyCharmu: View → Quick Documentation
- v konzoli Pythonu: `help(jméno_funkce)`
  - konzole v PyCharmu: Tools → Python Console

Dokumentace na webu

- `https://docs.python.org/3/`
- Tutorial, Language Reference, *Library Reference*

Vstup a výstup

Výstup

- funkce `print()`
- libovolné množství parametrů
- parametry odděleny mezerou
  - dá se změnit parametrem `sep`
- za posledním parametrem konec řádku
  - dá se změnit parametrem `end`
  - `end=""` způsobí, že výpis nebude ukončen koncem řádku
  - znak konce řádku v řetězci zapisujeme jako `\n`

```
print("Dnes", "je", "pěkně")
print("Dnes", "je", "pěkně", sep=":")
print("Dnes", "je", "pěkně", end="")
print("Tohle nebude na dalším řádku.")
```

Vstup a výstup

Vstup

- funkce input()
- dá se volat bez parametrů nebo s jedním parametrem (výzvou)
- vrací řetězec; ten je možno např. uložit do proměnné

```
name = input("What is your name? ")
print("Hello, ", name, ", nice to meet you.", sep="")
```

Co když budeme chtít načíst číslo?

```
x = input()
y = input()
print(x + y)  # co se stane?
```

Typy

- proměnné v Pythonu nemají typ samy o sobě (mohou držet libovolné hodnoty)
- ale hodnoty v Pythonu *mají* typ

Základní typy v Pythonu

- číselné
  - celá čísla: `int`
  - „reálná“ čísla: `float`
  - komplexní čísla: `complex`
- řetězce: `str`
- pravdivostní hodnoty: `bool`
- seznam (pole)
- ntice
- slovník
- ...

Typy

Zjištění typu

- funkce `type()`
- vrací speciální objekt, který se dá porovnávat se jménem typu

```
print(type(3))
print(type(3.0))
print(type("3"))
print(type(1 + 1 == 2))

x = 3.14
if type(x) == float:
    print("x je reálné číslo")
```

Typy

Převody mezi typy

- implicitní (dějí se automaticky):
  - celé číslo → reálné číslo
  - pravdivostní hodnota → číslo
- explicitní (je třeba si o ně říct):
  - provádí se pomocí funkce, která se jmenuje stejně jako cílový typ

```
# výška mostu podle doby volného pádu
height = float(input("Doba v sekundách: "))
print("Výška v metrech:", 9.81 * height * height / 2)
```

[ukázky převodu řetězec ↔ číslo]

Výrazy a operace

Výrazy

- kombinace proměnných a konstant pomocí operátorů

Operace

- aritmetické: sčítání, násobení, ...
- porovnávací: `==`, `<`, `<=`, ...
- logické: `and`, `or`, `not`, ...
- spojování řetězců
- ...

- priorita operátorů, pořadí vyhodnocování

Operace

Aritmetické operace

- sčítání `+`, odčítání `-`
- násobení `*`
- dělení `/` (výsledkem je vždy číslo typu `float`
- dělení se zaokrouhlením dolů `//`
- zbytek po dělení `%`
- umocňování `**`
  - `x ** 10` je „x na desátou“

- zkrácený zápis: `x += 5` znamená `x = x + 5`
  (a podobně pro ostatní operace)

Operace

Porovnávání

- rovnost `==`, nerovnost `!=`, „menší než“ `<`, „menší nebo rovno“ `<=`, ...
- v Pythonu se dají i řetězit: `1 < x < 7`
  - specialita Pythonu, v jiných jazycích nemusí fungovat
- porovnávání řetězců je *lexikografické* (tj. jako ve slovníku)

Logické operace

- konjunkce **`and`**, disjunkce **`or`**, negace **`not`**
- zkrácené vyhodnocování `and` a `or`
  - vyhodnocuje se zleva, vyhodnocování končí, jakmile je jasný výsledek: `1 + 1 == 2 or x == 3`

Operace s řetězci

- spojování řetězců `+`: `"ahoj" + "lidi" == "ahojlidi"`
- opakování řetězce `*`: `"ahoj" * 3 == "ahojahojahoj"`

Pořadí vyhodnocování

- většinou „intuitivní“
  - operace se vyhodnocují zleva doprava
  - `*` má přednost před `+` apod.
- pokud jste na pochybách
  - podívejte se do dokumentace
  - použijte závorky

`https://docs.python.org/3/reference/expressions.html#operator-precedence`

Cyklus `for`

- minule jsme viděli: `for i in range(10):`
- `for` v Pythonu je obecnější
  - `range()` je jeden z druhů objektů, kterými se dá procházet
  - dalšími jsou řetězce, seznamy, ntice, … (uvidíme dále)

`range`

- základní verze `range(n)`: postupně procházíme čísla od 0 do `n - 1`
- `range(from, to)`: procházíme čísla od `from` do `to - 1`
- `range(from, to, step)`: procházíme čísla od `from`, velikost jednoho kroku je `step`, skončíme na posledním čísle před `to`

```
for i in range(10, 100, 7):
    print(i)
```

- jak můžeme počítat pozpátku? použijeme záporný krok

Cyklus `while`

Jak napsat cyklus **`for`** *pomocí cyklu* **`while`***?*

```
for i in range(10, 100, 7):
    print(i)

i = 10
while i < 100:
    print(i)
    i += 7
```

- k zamyšlení: mezi těmito dvěma cykly je drobný rozdíl, v čem?
  - jaká je hodnota proměnné `i` po skončení cyklů?

Příklad: Faktoriál pomocí cyklu

```
n = int(input("Zadej číslo: "))

f = 1

for i in range(1, n + 1):
    f *= i

print("Faktoriál čísla", n, "je", f)
```

- jak odkrokovat program?
  - pomocné výpisy
  - debug ve vývojovém prostředí (později)
  - `http://pythontutor.com/`

Jak byste napsali faktoriál pomocí cyklu ***`while`****?*

Příklad: Převod do binárního zápisu

Lidé se dělí do 10 skupin:
na ty, co rozumí
binárnímu zápisu,
a na ty, co mu nerozumí.

Příklad: Převod do binárního zápisu

vstup: číslo v desítkové soustavě
výstup: číslo ve dvojkové soustavě

Jak převedeme číslo do dvojkové soustavy?

```
n = int(input())
output = ""

while n > 0:
    if n % 2 == 0:
        output = "0" + output
    else:
        output = "1" + output
    n //= 2

print(output)
```

Další příkazy pro řízení cyklů

(pokročilejší)

- `break`: ukončí provádění cyklu
- `continue`: ukončí aktuální iteraci cyklu a přejde na další
- specialita Pythonu – `else` větev u cyklu: provede se na konci cyklu, pokud cyklus nebyl ukončen pomocí `break`
  - v tomto předmětu nebudeme používat

Struktura programu

Sekvenční řazení příkazů

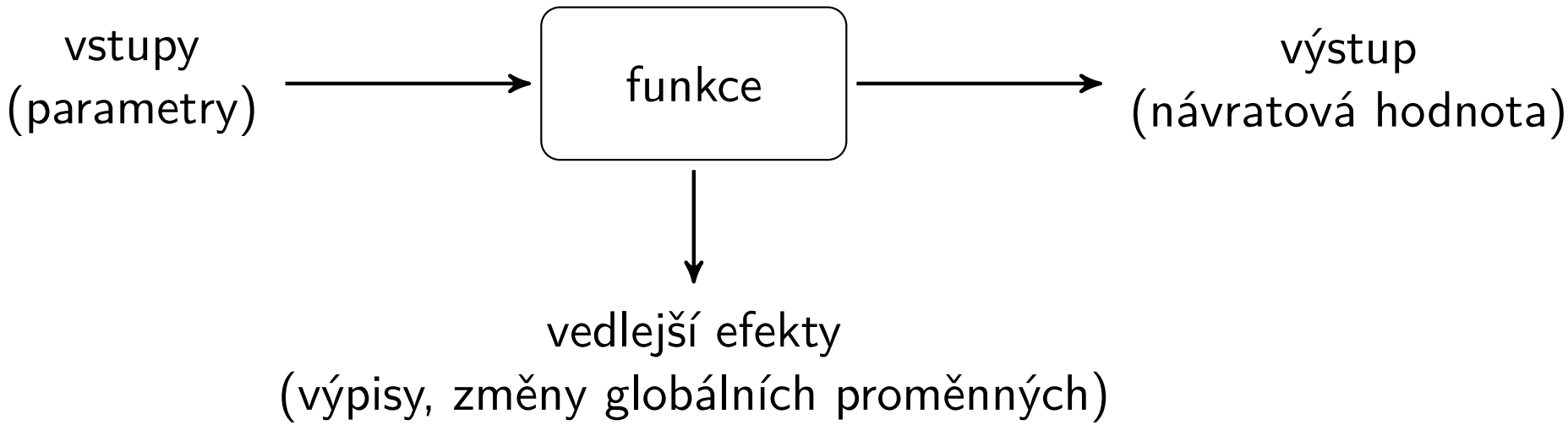
- pod sebe, jeden na řádek
- více příkazů na jednom řádku můžeme oddělit ;
  - nedoporučeno používat
- příliš dlouhý řádek můžeme rozdělit na více řádků
  - pokračování řádku naznačíme znakem \ na konci řádku

Zanořování podmínek/cyklů

- tělo odsazujeme, doporučeny 4 mezery
- můžeme vzájemně zanořovat
  - podmínka uvnitř cyklu
  - cyklus uvnitř cyklu
  - ...
  - libovolný počet zanoření (ale ...)

Funkce

- programy dělíme do funkcí
- funkce můžeme dále rovněž dělit do (menších) funkcí
- důvody už jsme si řekli minule



Funkce

- *vstup*: parametry funkce
- *výstup*: návratová hodnota
  - předáváme pomocí **return**
  - funkci můžeme použít kdekoli na místě výrazu

```
def fact(n):
    f = 1
    while n > 0:
        f *= n
        n -= 1
    return f

print("Faktoriál čísla 42 je", fact(42))
```

Příklad: Převod do binárního zápisu

```
def to_binary(n):
    output = ""
    while n > 0:
        if n % 2 == 0:
            output = "0" + output
        else:
            output = "1" + output
        n //= 2
    return output

print(to_binary(22))
```

Rozdíl mezi výstupem a vracením

`print` vs. `return`

- jaký je rozdíl mezi těmito funkcemi?

```
def funr(n):
    return n + 17

def funp(n):
    print(n + 17)
```

- funkce `funr` vrátí hodnotu
  - můžeme ji uložit do proměnné `x = funr(10)`
  - můžeme ji použít ve výrazu `print(20 * funr(10))`
- funkce `funp` vytiskne na výstup hodnotu
  - a nevrací nic

Volání funkcí

Vnořené volání funkcí

- funkce mohou volat další funkce
- po dokončení vnořené funkce se program vrátí do místa, odkud se funkce zavolala
- funkce může dokonce volat sebe samu
  - tzv. *rekurze*
  - pokročilejší téma, v tomto předmětu nebudeme potřebovat

Lokální proměnné

- proměnné definované uvnitř funkce jsou v Pythonu lokální
- každé volání funkce má své vlastní proměnné
- existují i globální proměnné
  - o těch se dozvíme později
  - doporučení používat co nejméně

Příklad: lokální proměnné

```
def f(n):
    x = 2 * n
    print("f: x =", x, "n =", n)

def g(n):
    x = 3 * n
    print("g před f: x =", x, "n =", n)
    f(n + 7)
    print("g po f: x =", x, "n =", n)

x = 17
n = 33
print("hlavní program před g: x =", x, "n =", n)
g(9)
print("hlavní program po g: x =", x, "n =", n)
```

Vnořené volání: jednoduchý příklad

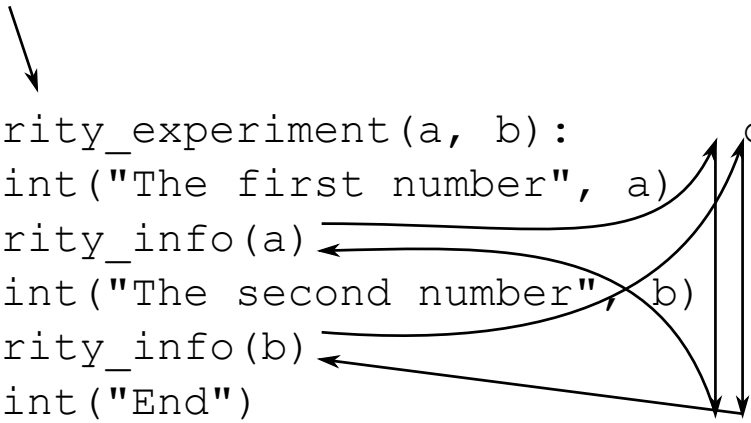
```
def parity_info(number):
    print("Number", number, end=" ")
    if number % 2 == 0:
        print("is even")
    else:
        print("is odd")

def parity_experiment(a, b):
    print("The first number", a)
    parity_info(a)
    print("The second number", b)
    parity_info(b)
    print("End")

parity_experiment(3, 18)
```

Vnořené volání: jednoduchý příklad

```
parity_experiment(3, 18)
```



```
def parity_experiment(a, b):
    print("The first number", a)
    parity_info(a)
    print("The second number", b)
    parity_info(b)
    print("End")
```

```
def parity_info(number):
    print("Number", number, end=" ")
    if number % 2 == 0:
        print("is even")
    else:
        print("is odd")
```

Funkce: Speciality Pythonu

(vyskytují se i v některých jiných jazycích)

- implicitní hodnoty parametrů
- volání pomocí jmen parametrů

```
def fun(x, y=3):
    print("x =", x, "y =", y)

fun(1, 2)
fun(1)
fun(y=7, x=4)
```

- libovolný počet parametrů a další speciality

Doporučení pro psaní funkcí

- nejprve si ujasnit *specifikaci*
  - jaké potřebuje funkce vstupy?
  - co bude výstupem funkce?
- funkce by měly být *krátké*
  - „jedna myšlenka“ na jednu funkci
  - funkce by se měla vejít na jednu obrazovku
  - jen pár úrovní zanoření
- co když je funkce příliš dlouhá?
  - najít v ní menší logické celky
  - rozdělit na menší funkce

Příklad: výpis šachovnice

```
#.#.#.#.
.#.#.#.#
#.#.#.#.
.#.#.#.#
#.#.#.#.
.#.#.#.#
#.#.#.#.
.#.#.#.#
```

- jak?

Příklad: řešení hádanky hrubou silou

Hádanka: Farmář chová prasata a slepice. Celkem je na dvoře 20 hlav a 56 noh. Kolik má slepic a kolik prasat?

- jak vyřešit pro konkrétní zadání?
- jak vyřešit pro obecné zadání (`h` hlav a `n` noh)?

Možné přístupy:

- „chytré řešení“: řešení systému lineárních rovnic
- „hrubou silou“: vyzkoušíme všechny možnosti

K zamyšlení: Co kdyby farmář ještě navíc choval osminohé pavouky?

Zásady čitelnosti kódu

Obecné

- pište tak, aby byl jasný záměr
  - rozumné pojmenování proměnných, funkcí, apod.
  - v případě komplikovanějšího kódu dokumentace (komentáře)
  - ale nejlepší kód je samodokumentující se kód
- větší kód rozčleňte do rozumných menších celků
  - funkce, (objekty, moduly, ...)

Konkrétní

- různé jazyky mají různou představu o „slušném“ kódu
- různé skupiny (různá pracovní prostředí) mají různé zvyky
  - coding styles, coding guides
- pro Python existuje oficiální doporučení: PEP8

PEP8

Kde najít, jak použít?

- `https://www.python.org/dev/peps/pep-0008/`
- PyCharm umí kontrolovat sám [ukázka]
- jiné nástroje: `pep8`, `autopep8`, `flake8` (příkazový řádek)

Důležité body

- zarovnávání kódu; max. délka řádku 79 znaků; dva prázdné řádky mezi funkcemi
- mezery kolem operátorů; žádné mezery za otevírací (před uzavírací) závorkou
  - výjimka: žádné mezery kolem = u parametrů funkcí
- pojmenování věcí
  - proměnné, funkce: `lower_case_with_underscores`
  - konstanty: `UPPER_CASE_WITH_UNDERSCORES`
  - třídy (později): `CamelCase`

Závěrečný kvíz

`https://kahoot.it`

Závěr

Co jsme se dnes dozvěděli?

- `print`: oddělovač a ukončení umíme změnit
- vstup pomocí funkce `input`
- základní typy v Pythonu; jak je zjistit, jak přetypovat
- základní operace: aritmetické, logické, řetězcové, porovnávání
- vztah cyklů `for` a `while`
- funkce s návratovou hodnotou; lokální proměnné
- kde najít a jak dodržet zásady čitelnosti kódu

Co bude příště?

- věci, které jsme dnes pořádně nestihli
- programy, pracující s čísly
- ladění programu (debugging)
- náhodná čísla, simulace