České vysoké učení technické v Praze FIT

# Programování v Pythonu

Jiří Znamenáček

*Příprava studijního programu Informatika je podporována projektem financovaným z Evropského sociálního fondu a rozpočtu hlavního města Prahy.*

*Praha & EU: Investujeme do vaší budoucnosti*

# Python – Úvod do Python'u

## Historie Pythonu


- Guido van Rossum (BDFL), přelom 80. a 90. let, následovník jazyka ABC
- 16. 10. 2000 – verze 2.0 (komunitní vývoj, *full garbage collector*, prvotní podpora pro Unicode...)
- 3. 12. 2008 – verze 3.0, záměrně první zpětně nekompatibilní (skutečná podpora Unicode'u, sjednocení syntaxe a „vnitřností“, vyčištění systémové knihovny...)

Řada 2.x tu bude ještě dlouho (externí knihovny), řada 3.x je ale v mnoha ohledech „hezčí“ jazyk (a zatím prochází dosti překotným vývojem).

## Python 2.x versus Python 3.x

- nejviditelnější změna
  `print 'Ahoj, Karle!'` VS. `print('Ahoj, Karle!')`
- nejdůkladnější změna
  `u"Ahoj, světe!"` VS. `"Ahoj, světe!"`
- sjednocení syntaxe a „vnitřností“
- vyčištění systémové knihovny
- atd.

## Užitečné odkazy

- http://python.org
- http://python.org/doc/

- http://cheeseshop.python.org/pypi
- http://code.activestate.com/recipes/langs/python/
- http://DiveIntoPython.org
- http://DiveIntoPython3.org

# Charakter jazyka

Python je

- interpretovaný, dynamicky typovaný high-level jazyk
- procedurální, ale i objektový a dokonce i funkcionální

Dále též

- všechno v Pythonu je objekt, navíc dokonce „first-class object“ (tj. je možné cokoliv předat jako argument do funkce; tedy samozřejmě třeba i jinou funkci!)
  `"Ahoj, světe!".split()`
- proměnné se nedeklarují, rovnou se jim přiřazuje hodnota (ale dosud nepoužitou proměnnou nemůžete „zavolat“, dostali byste `NameException`)
- case-sensitive (a má odpovídající konvence na psaní názvů jednotlivých typů objektů)
  `joined_lower` - funkce & metody & atributy, `ALL_CAPS` - konstanty, `StudlyCaps` - třídy

# Poznámka ke kódování

```
# -*- coding: utf-8 -*-
# encoding: utf-8
```

- v historicky dávné době byl Python asi čistě ASCII (nevím, nezažil :)

- v Pythonu 2.x se dá určit kódování souboru, ale řetězce jsou brány jako sekvence osmibitových znaků
  `"ASCII string" VS. u"Ahoj, světe!"`
- až v Pythonu 3.x je konečně všechno nativně UTF-8 (a to dokonce až na úroveň názvů proměnných či funkcí)
  `"Ahoj, světe!" VS. b"bytes"`

Ale spousta nástrojů s tím ještě nepočítá!

# Klíčová slova

```
False       class       finally     is          return
None        continue    for         lambda      try
True        def         from        nonlocal    while
and         del         global      not         with
as          elif        if          or          yield
assert      else        import      pass
break       except      in          raise
```

```
>>> help()
>>> help('del')
```

# Comments & docstrings

```
# funkce pro zpracování vstupního čísla
def fnce(číslo):
  """
  Funkce fnce() očekává na vstupu právě jedno číslo.
  Na svém výstupu vrací toto číslo prohnané modulem zpracovator.
  """
  import zpracovator
  n = zpracovator.zpracuj()
  return n
```

→ *docstring* : jak kód používat
→ *comment* : jak a proč daný kód funguje (je napsán)

# Datové typy

| **proměnné (*mutable*)** | **neproměnné (*immutable*)** |
|---|---|

| **proměnné (*mutable*)** | **neproměnné (*immutable*)** |
|---|---|
| — | Strings |
| Lists | Tuples |
| Sets | Frozen Sets |
| Byte Arrays | Bytes |
| Dictionaries | — |

**Číselné:** int, float, complex

**Sekvenční:** str, list, tuple, range, bytes, bytearray

**Množinové:** set, frozenset

**Mapovací:** dict

# Příklady (datové typy)

- **Boolean** – `True`, `False`
- **Number** – `1`, `2.3`, `4/5`, `2+3j`
- **String** – `"Ahoj, světe!"`
- **Byte** – `b""`
- **List** – `[2, 'a', {},]`
- **Tuple** – `(2, 'a', {},)`
- **Set** – `{2, 'a', {},}`
- **Dictionary** – `{ 'jmeno': ('Karel', 'Novák'), 'vek': 23, }`

# Operátory

- číselné: `+ - * ** / // %`
- logické: `and or not`
- porovnávací: `< > == != <= >=`

- bitové: `<<` `>>` `&` `|` `^` `~`
- speciální: `in` , `not in` ; `is` , `is not`

Občas je možné si výraz zkrátit: `a = a + 3` je totéž jako `a += 3`

Pozor na priority (a typy porovnávaných objektů)! Už z důvodu čitelnosti je lepší závorkovat.

# Příklady (typy, operátory, porovnávání)

### Čísla:

```
>>> a = 5
>>> b = 3
>>> a + b
8
>>> type(5)
<class 'int'>
>>> a += 4
>>> a
9

>>> c = 1.5
>>> c
1.5
>>> type(c+a)
<class 'float'>
>>> 5/2
2.5
>>> 5//2
2
>>> 5**2
25
```

### Řetězce:

```
>>> s = "ahoj"
>>> type(s)
<class 'str'>
>>> s + " světe"
'ahoj světe'
>>> len(s)
4
>>> s[0]
'a'
>>> s[-1]
'j'
```

### Přiřazení a porovnání:

```
>>> a = b = 0
>>> a
0
>>> b
0
>>> a == b
True
>>> type(3 == 5)
<class 'bool'>
>>> "bla" == 'bla'
True
>>> (1, 2) == (1, 2)
True
>>> (1, 2) == (1, 2, )
True
>>> 1, 2 == 1, 2
(1, False, 2)
```

### `is` versus `is not`:

```
>>> a = 1
>>> b = 1
>>> a == b
True
>>> a is b
True

>>> a = [1]
>>> b = [1]
>>> a == b
True
>>> a is b
False
>>> a is not b
True
```

# Přiřazení (či spíše „pojmenovávání“)

```
>>> xs = ['a', 'h', 'o', 'j']
>>> ys = xs
>>> ys
['a', 'h', 'o', 'j']

>>> xs.remove('o')
>>> xs
['a', 'h', 'j']
>>> ys
['a', 'h', 'j']
```

Proměnné (*mutable*) typy jsou v Python'u předávány odkazem, neproměnné (*immutable*) prozměnu zase hodnotou! Na začátku si na tom asi párkrát nabijete nos, ale časem na tuhle nakonec celkem logickou podivnost (zvanou, pokud vůbec nějak, *předávání objektem*) zvyknete.

# Základní řídící konstrukce

```
v1, v2 = 1.3, "ahoj"
del v1
```

```
if PODMINKA:
    BLOK
```

```
for i in SEKVENCE:
    BLOK
```

```
while PODMINKA:
    BLOK
```

Všimněte si použití odsazování („rozpoznávací znak Python'u“ :-) jako oddělovače bloků!

# Příklady (řídící konstrukce)

### Cyklus `for` a `range()`:

```
>>> for word in ["welcome", "to", "python"]:
...     print(word, end=" ")
...
welcome to python >>>

>>> sum = 0
>>> for i in range(10):
...     sum += i
...
>>> sum
45

>>> range(5)
range(0, 5)
>>> type( range(5) )
<class 'range'>
>>> list( range(5) )
[0, 1, 2, 3, 4]

>>> range(4,6)
range(4, 6)
>>> list( range(4,6) )
[4, 5]

>>> range(1,7,2)
range(1, 7, 2)
>>> list( range(1,7,2) )
[1, 3, 5]
```

### Cyklus `while`:

```
>>> while True:
...     print( "ahoj" )
...     break
...
ahoj

>>> i = 0
>>> while i < 5:
...     print(i)
...     i += 1
...
0
1
2
3
4
```

### Rozhodování `if – elif – else` a podmínky:

```
>>> if 4 == 5:
...     print('foo')
... else:
...     print('bar')
...
bar

>>> False and False or True
True
>>> not True
False

>>> a = "foo"
>>> if a in ['blue', 'yellow', 'red']:
...     print( a + " is a color." )
... elif a in ['US', 'China']:
...     print( a + " is a country." )
... else:
...     print( "I don't know what " + a + " is!" )
...
I don't know what foo is!
```

# Sekvence

Konečné uspořádané množiny prvků indexovaných nezápornými čísly. Sdílí mnoho vlastností, metod apod.:

- `len(xs)` – vrací délku příslušné sekvence `xs`
- `xs[i]` – vrací prvek sekvence na pozici `i` (smysl má i výraz typu `xs[-2]`)
- `xs[i:j]` – vrací podsekvenci od indexu `i` po index `j-1` (smysl mají i výrazy typu `xs[:3]`, `xs[-2:]` apod., někde též `xs[i:j:k]`)
- `zs = xs + ys` – spojování sekvencí
- `for x in xs:`
  smyčka přes všechny prvky sekvence
- `x in xs` – zjištění výskytu prvku v sekvenci

Pár příkladů na řetězcích:

```
>>> s = "Ahoj, světe!"

>>> len(s)
12

>>> s[0]
'A'
>>> s[-2]
'e'
>>> s[2:5]
'oj,'
>>> s[:5]
'Ahoj,'
>>> s[-4:]
'ěte!'
>>> s[2:9:3]
'o ě'

>>> for c in s:
...     print(c, end=' ')
...
A h o j ,   s v ě t e ! >>>

>>> 'A' in s
True
>>> 'a' in s
False
```

# Triky s přiřazováním

Typicky pythonovské prohození hodnot dvou proměnných:

```
a, b = b, a
```

Přiřazení blíže neurčeného počtu dat do menšího počtu proměnných:

```
>>> a, *b, c = 'první', 1, 2, 'ahoj', 'poslední'
>>> a
'první'
>>> b
[1, 2, 'ahoj']
>>> c
'poslední'
```

# Ukázka – QuickSort

```
def qs(a):
    if a == []:
        return []
    else:
        pivot = a[0]
        left  = [x for x in a      if x <  pivot]
        right = [x for x in a[1:] if x >= pivot]
        return qs(left) + [pivot] + qs(right)
```

# Ukázka - faktoriál

Klasicky pomocí rekurze:

```
def faktorial(n):
    if n == 0:
        return 1
    else:
        return n * faktorial(n-1)

print( faktorial(9) )
```

Méně přehledně, ale také podstatně méně náročně pomocí cyklu:

```
n = 9
out = 1

while n:
    out *= n
    n -= 1

print( out )
```

Líně pomocí knihovní metody:

```
import math

# Return x factorial. Raises ValueError if x is not integral or is
print( math.factorial(9) )
```

# Ukázka - prvočísla

Hodně neklasicky, ale neoptimalizovaně:

```
>>> for n in range(100):
...     for x in range(2,n):
...         if n % x == 0:
...             break
...     else:
...         print( n, end=' ')
...
0 1 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79
```

Poznámka: Větev s **else** u cyklu *for* se provede pouze tehdy, je-li for-cyklus ukončen normálně, tedy nikoli právě na *break*.

# Odbočka I - print()

Pravděpodobně nejpoužívanější funkce Python'u má syntaxi:

```
print( [object, ...]
       [, sep=' '][, end='\n'][, file=sys.stdout] )
```

Příklad:

```
>>> s1, s2, s3 = 'ahoj', 'světe', 'jak se máš'
>>> print(s1, s2, s3, sep=' | ', end='\n')
ahoj | světe | jak se máš
```

# Odbočka II - pprint()

```
>>> tup = ('spam', ('eggs', ('lumberjack',
            ('knights', ))))
>>> stuff = ['a' * 10, tup, ['a' * 10, 'b' * 10],
             ['c' * 8, 'd' * 8]]

>>> import pprint
>>> pprint.pprint( stuff, width=40 )
['aaaaaaaaaa',
 ('spam',
  ('eggs',
   ('lumberjack', ('knights',)))),
 ['aaaaaaaaaa', 'bbbbbbbbbb'],
 ['cccccccc', 'dddddddd']]
```

→ Mírně zjednodušený příklad přímo z dokumentace k Python'u.

# Závěrečná odbočka - dir()

Snad nejužitečnější funkce v Python'u:

```
dir( OBJEKT )
```

Příklad:

```
>>> dir( 'ahoj' )
['__add__', '__class__', '__contains__', '__delattr__', '__doc__',
```

Typ *řetězec* má tedy následující atributy:

- „magické“:

```
__add__
__class__
__contains__
__delattr__
__doc__
__eq__
__ge__
__getattribute__
__getitem__
__getnewargs__
__getslice__
__gt__
__hash__
__init__
__le__
__len__
__lt__
__mod__
__mul__
__ne__
__new__
__reduce__
__reduce_ex__
__repr__
__rmod__
__rmul__
__setattr__
__str__
```

- „běžné“:

```
capitalize
center
count
decode
encode
endswith
expandtabs
find
index
isalnum
isalpha
isdigit
islower
isspace
istitle
isupper
join
ljust
lower
lstrip
partition
replace
rfind
rindex
rjust
rpartition
rsplit
rstrip
split
splitlines
startswith
strip
swapcase
title
translate
upper
zfill
```