



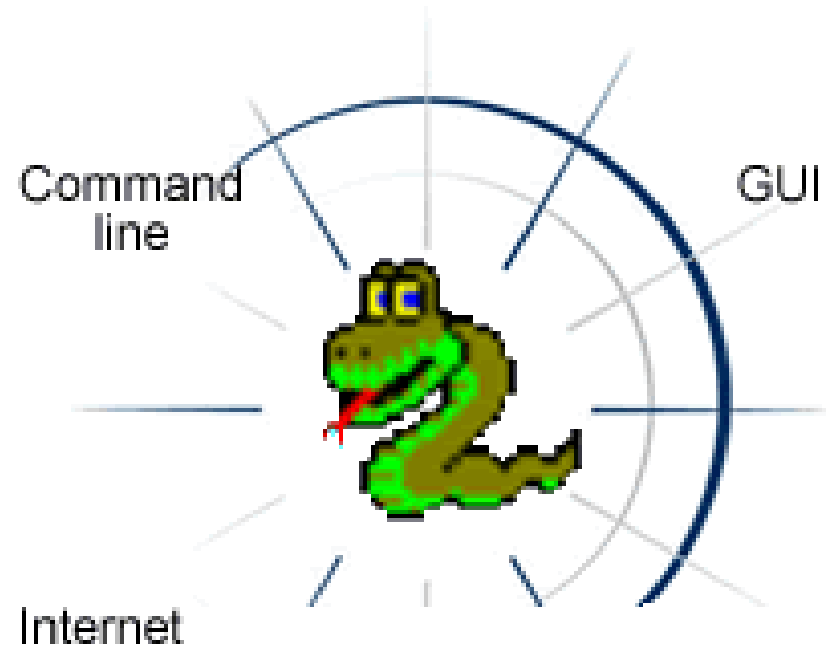
Python

Klára Pešková

peskova@braille.mff.cuni.cz

Jaký je Python?

- rychlý vývoj aplikací (včetně GUI)
 - nezávislý na platformě (Windows, Unix, Macintosh)
- přenositelnost
- ...a je zadarmo!
(open source)



Programovací jazyky

■ Strojové

- instrukce "1:1"
- Assembler

■ Systémové

- "high-level": primitiva jazyka nekorrespondují přímo se strojovými instrukcemi
- dá se v nich napsat "všechno"
- C, Pascal, Java...

■ Skriptovací

- ještě víc "high-level" - "glue", lepíme z komponent
- Python, Perl, bash, Visual Basic, ...

Rychlost

■ Systémové

- rychleji to běží
- pomaleji se to píše / učí se

■ Skriptovací

- rychleji se to píše / učí se
- pomaleji to běží

Skriptovací jazyky - typické vlastnosti

- interpretované
- původně spíše special-purpose
- reflexe (kód = data)
- většinou bez explicitně určovaných typů
 - typy určeny až za překladu
- často podpora zpracování textu

Kdy použít a kdy ne?

- Rychlost vývoje je kritická
 - Prototypování
 - Předpoklad požadavků na změny
 - "Special-purpose"
 - GUI, řetězce, ...
 - Nižší nároky na programátora
- Rychlost programu je kritická
 - "Low-level" kód
 - ovladač karty
 - Manipulace s velkými daty
 - Kritické aplikace, silná potřeba ladění

Python - úvod

- Vznik – 1991, Python 3.0 – 3. prosince 2008
- „krásný, jednoznačný, jednoduchý“
- Hybridní (víceparadigmatický jazyk) – objektové i procedurální programování, prvky funkcionálního programování
- Open source
- Skriptovací, interpretovaný jazyk (jako např. PHP, JavaScript, Perl)
- pomalý x rychlý vývoj aplikací
- Implementovaný v jazyce C (výkonově kritické knihovny rychlé)

Úvod II.

- snadné začátky - interaktivní režim
- vhodný pro výuku programování, expresivní
- vhodný pro krátké skripty i větší projekty
- automatická správa paměti
- konzistentní vzhled programů
- dobrá podpora OOP
- jednoduché propojení s jinými programovacími jazyky (C/C++, Java, COM objekty)



Guido van Rossum – tvůrce Pythonu

BDFL - Benevolent dictator for life

...he continues to oversee Python development, making decisions when necessary.

Používá se?

- prototypování
- umělá inteligence (BioPython, matplotlib, NumPy, Scikitlearn)
- webové aplikace (Django, Zope)
- skriptování - např. Gimp, Blender, Corel Paint Shop Pro, Inkscape, LibreOffice
- Wikipedia, Google, Yahoo!, CERN, NASA, Facebook, Amazon, Instagram, Spotify
- The social news networking site Reddit is written entirely in Python.

zdroj: wikipedia

TIOBE index (<https://www.tiobe.com/tiobe-index/>)

Python vs. C, C++, Pascal

- dynamické typování
- automatické řízení paměti
- kratší a přehlednější kód, snadnější vývoj
- ale pomalejší a větší paměťové nároky
 - C 3-5x rychlejší
 - matematické výpočty: C až 10x rychlejší
 - práce s řetězci - stejně (dobré knihovny)
- řešení: možnost psát kritické moduly v C/C++

Syntax

- jeden z cílů při vývoji - srozumitelný, krátký kód
- odsazování
- nutí psát přehledný kód; jednotý vzhled
- dynamické typování
- case sensitive

```
def f():  
    if random.random() < 0.5:  
        return False  
    else:  
        return True
```

```
f()
```

Základní číselné typy

- Celá čísla - int

`a = 5 long (C)`

- Reálná čísla – float

`b = 5.5; c = -3e-3 double (C)`

- Dlouhá celá čísla (neomezeně dlouhá) - long

`c = 200000000000000000000000000000004432000000L`

- Komplexní čísla - complex

`x = (3+3j)`

`x.real`

`x.imag`

Základní číselné typy II.

- Aritmetické operátory:

- `+`, `-`, `*`, `/`, `//`, `**`, `%`

- Matematické funkce

- `round()`

- `abs()`

- `max()`, `min()`

- další funkce v modulu `math` (`cmath`):

- `ceil()`, `floor()`

- Goniometrické funkce, logaritmus...

Znakové řetězce - string

- `s = "Ahoj"`

- k položkám řetězce je možné přistupovat, ale není možné je měnit

```
>>> s[0] # indexování od 0  
'A'
```

```
>>> s[1] = 'a'
```

- délka řetězce - funkce `len(t)`

- nativní podpora Unicode

Práce se znakovými řetězci

■ spojování řetězců

```
>>> s = "Ahoj"  
>>> r = "lidi"  
>>> print (s+" "+r)  
'Ahoj lidi'
```

■ "násobení" řetězců

```
>>> s = "ra"+"ta"*2  
>>> s  
'ratata'
```

■ slicing

```
>>> t[1:4] # vypise znaky <1,4)  
>>> t[:2] # prvni dva znaky  
>>> t[2:] # vse krome prvnich dvou znaku  
>>> t[-1] # vypise posledni znak
```

Práce se znakovými řetězci II.

■ některé vestavěné funkce pro práci s řetězci

- `capitalize()`
- `center(width[, fillchar])`
- `find(sub[, start[, end]])` # vraci index
- `isalnum()`, `isalpha()`, `isdigit()`, `islower()`, `isspace()`
- `join(seq)`
- `strip([chars])`
- `split([sep [,maxsplit]])`, `splitlines([keepends])`
- `replace(old, new[, count])`

■ modul string

- `whitespace` - řetězec znaků
- `digits`, `hexdigits`

■ formátování řetězců

```
>>> "{0} je {1} {2}".format("Python", "cool", "jazyk")
Python je cool jazyk.
```


Seznamy - list

- `s1 = [1, 2, 3, 4]`
- položky seznamu mohou být různé datové typy (i seznamy, n-tice...)

```
s2 = [a, 2, "tri", [4, "ctyri"], 5.5]
```

- k položkám je možné přistupovat i je měnit

```
>>> s1[0]
```

```
1
```

```
>>> s2[3]
```

```
[4, 'ctyri']
```

```
>>> s2[3][1]
```

```
'ctyri'
```

```
>>> s2[3] = 'slunicko'
```

```
>>> s2
```

```
[a, 2, 'tri', 'slunicko' , 5.5]
```

Seznamy II.

- seznamy je možné spojovat

```
>>> s3 = s1 + s2 + [5]
```

```
# všechny položky musí být seznamy
```

- funkce

- `len(s1)` - vrátí počet položek seznamu

- při práci se seznamy se kopíruje pouze ukazatel na seznam
- modul `copy` - `copy.deepcopy(seznam)`

Seznamy III.

■ Seznam jako zásobník

```
>>> zasobnik = [3, 4, 5]
>>> zasobnik.append(6) # přidá prvek na konec
>>> zasobnik.pop()    # vrátí prvek z konce
                        # zásobníku a smaže ho
```

■ Seznam jako fronta

```
>>> fronta = ['Jan', 'Pavel', 'Tomas']
>>> fronta.append(3)
>>> fronta.pop(0)     # vrátí první prvek a
                        # smaže ho
```

Množiny (set)

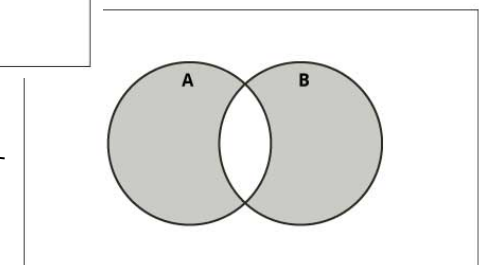
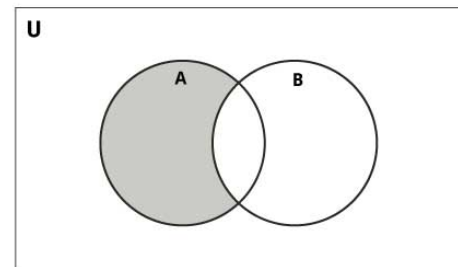
- neuspořádaný datový typ
- set, frozenset (neměnné)
- podpora množinových operací – operátory:

| sjednocení

& průnik

- rozdíl

^ symetrický rozdíl



N-tice - tuples

- prvky n-tice není možné měnit => méně místa v paměti, rychlejší přístup

- použití - jako klíče slovníků

- mají neměnnou velikost, neměnné prvky

```
>>> n = (1, 'A', 3)
```

- jednoprvková n-tice:

```
>>> a = ('a',)
```

- není nutné psát závorky

```
>>> i, j, k = n
```

```
>>> i, j = j, i # prohodí prvky
```

... další věci

- `list(n)` - udělá z n-tice, řetězce seznam
- `tuple(n)` - vytvoří n-tici ze seznamu, z řetězce

- k proměnné lze přiřadit jakýkoliv typ
- hodota `None`
- Boolean
 - `True, False`
 - hodnota `false` - 0, prázdná hodnota
 - `true` - všechno ostatní

Slovník - dictionary

- = "hash", "asociativní pole", pole indexované čímkoliv
- klíče mohou být neměnné datové typy - čísla, znakové řetězce, n-tice
- položky nejsou uspořádané

```
vzdalenosti = {'Karlův most': 160, 'hospoda': 50,  
              'Hrad': 300, 'zachody': 'poblíž' }
```

- změna položky:

```
vzdalenosti['zachody'] = "v prvním patře"
```

- přidání položky:

```
vzdalenosti['koupalště'] = '1 km'
```

- smazání položky:

```
del vzdalenosti['zachody']
```

Slovník II.

- vypsání klíčů:

```
for klic in vzdalenosti:  
    print(klic)
```

- testování přítomnosti klíče:

```
'hospoda' in vzdalenosti # True, False  
vzdalenosti.get('potravinny', 'neznama')
```

- př. použití - řádká matice:

```
matice = {(0, 0): -3, (1, 2): 7,  
          (2, 0) : 4, (3, 2): 5}
```


Podmínky

- `if, else, elif`

- **syntax s dvojtečkou**

```
if i < 5:
```

```
    print ("i je mensi nez 5")
```

```
else:
```

```
    print ("i je vetsi nebo rovno nez 5")
```

- Python nemá "switch"

Operátory

- aritmetické:

- `+, -, *, /, //, **, %`

- logické:

- `and, or, not`

- porovnávací:

- `<, >, <=, >=, ==, !=, in, not in`

- `0 < a < 10`

Cykly - while

- while cyklus:

```
while i < 10:  
    print (i*3)  
    i += 1
```

- break, continue

- else

Cykly - for

- for cyklus - iterace přes skupinu položek (sekvenci) - seznam, n-tice, znakový řetězec...

```
for i in [1, 'deset', 100.5]:  
    print (i)
```

- funkce `range(-4, 6, 2)` – vrací iterátor ~ čísla z intervalu `<-4,6)` s krokem 2

```
for i in range(6):          # <0,6), tj.  
    print (i)              # provede se 6x
```

```
for i in range(-4, 6):     # krok 1  
    print (i)
```

Funkce

- proměnné ve funkcích jsou lokální

```
def dohromady(prvni, druhy) :  
    global x # globalni promenna  
    lokalni = prvni + druhy  
    return lokalni
```

```
>>> print (dohromady(1, 2))
```

```
>>> print (dohromady("lady", "bird"))
```

- Funkce v proměnné

```
>>> funkce = dohromady
```

```
>>> funkce(1, 2)
```

Funkce II. - předávání parametrů

- odkazem: seznam, slovník, instance třídy
- hodnotou: n-tice, řetězec, číslo
- různé způsoby předávání parametrů:
 - implicitní hodnota, předávání jménem: `arg=hodnota`
 - proměnlivý počet parametrů:
 - `*args # seznam`
 - `**args # slovník`

□ příklad:

```
def funkce(x, y=1, **dalsi):  
    pass  
funkce(2, y="3", prvni=1, druhe=2)
```

(Ne)měnné typy

- neměnné:

- čísla, znakové řetězce, n-tice

- "normální" kopírování
 - předávání hodnotou
 - klíče slovníků

- měnitelné:

- seznamy, slovníky, instance tříd

- kopíruje se pouze ukazatel
 - předávání odkazem

exec

■ Vykoná kód v řetězci

```
>>> a = 5
```

```
>>> s = "print ('hodnota a je:', a)"
```

```
>>> exec (s)
```

```
>>> s1 = "for i in [1, 2, 3] :\n"
```

```
>>> s2 = "\t print (i)\n"
```

```
>>> s = s1 + s2
```

```
>>> exec (s)
```


Užitečné funkce

- `len()`
- `int()` - přetypování
- `bin()`, `hex()`, `0b1001`, `0x00FF00`
- třídění:
 - merge sort
 - `sorted(x, reverse = True)`
 - `sorted(x, key = myKey)`

Práce se soubory

```
>>> f = open("pokus.txt", 'w')  
    otevře soubor pro zápis
```

```
>>> f.write("ahoj\n")  
    zapiše do souboru „ahoj“ a odřádkuje
```

```
>>> f.close()  
    zavře otevřený soubor
```

```
>>> f = open("pokus.txt")  
    otevře soubor pro čtení
```

```
>>> f.read()  
    načte celý soubor do jednoho řetězce
```

```
>>> f.readline()  
    načte jednu řádku
```

```
>>> f.readlines()  
    načte soubor po řádkách, řádky ukládá do seznamu
```

Python & OOP

- atributy třídy lze definovat za běhu (dokonce i nové třídy)

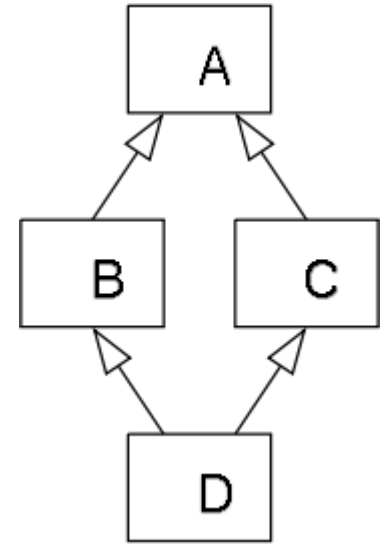
```
class Kruh:  
    pass  
k = Kruh()  
k.polomer = 7          # proměnná instance  
Kruh.barva = 'bila'   # proměnná třídy
```

```
>>> k.barva  
'bila'
```

- přístup k položkám `instance.promenna`
- pokud neexistuje proměnná instance, hledá se proměnná třídy (dá se využít pro nastavení implicitních hodnot)

OOP - dědičnost

```
class Kruh(Tvar):  
    def __init__(self):  
        # musíme explicitně zavolat konstruktore  
        # rodičovské třídy  
        super().__init__()
```



- pozor na kolize jmen proměnných => soukromé proměnné (`__promenna`)
- vícenásobná dědičnost (C++ omezení, Java zakazuje) - kolize názvů

OOP - správa paměti

- destruktory: `def __del__(self)`
- odstranění instance po smazání posledního ukazatele na ni

Poznámky:

- umí: přetěžování operátorů

List comprehensions

```
[expression for item in list if conditional]
```

```
for item in list:  
    if conditional:  
        expression
```

■ Př:

```
[x.upper() for x in ["a", "b", "c"]]
```

```
[x for x in range(10) if x%2==0]
```

Regulární výrazy

- modul re
- syntax:
 - . - libovolný znak (kromě dalšího řádku)
 - ^ - začátek řetězce
 - \$ - konec řetězce
 - * - nula nebo více opakování předešlého výrazu
 - + - jedno nebo více opakování předešlého výrazu
 - ? - nula nebo jedno opakování předešlého výrazu
 - [] - množina prvků, např. [a-zA-Z0-9]
 - [^W] - všechny znaky kromě W
 - {m} - m opakování výrazu
 - {m, n} - m až n opakování
- ```
>>> regexp = re.compile('[abc]+')
```
- vyhledávání regulárního výrazu v řetězci:
  - ```
regexp.search("ahoj") # vrací None nebo objekt
```
 - ```
regexp.findall("abeceda")
vrací seznam všech výskytů
```

# Co ještě...

- výjimky
- Tkinter
- dokumentační řetězce:  
 """ ... """  
 funkce.\_\_doc\_\_
- vytvoření .exe souboru (py2exe)



# Python 3000 vs Python 2.x

- `print` je funkce
  - `print x, => print(x, end=" ")`  
*# přidá na konec mezeru místo nového řádku*
- `range()`, `dict.keys()`, `dict.values()` **nevrací seznamy**
- `list.sort()` **nepodporuje argument `cmp`**
- `1/2` **vrací float; (`1 // 2`)**
- **podpora Unicode**
- `<>`, `!= => !=`
- `exec` **je funkce**

# Práce s moduly

- používání:
  - `import jmeno_modulu`
  - `from jmeno_modulu import jmeno_funkce`
  - `from jmeno_modulu import *`
- jak vypsat všechny dostupné moduly
  - `help()`, `modules`
  - v adresáři "Lib"
- `dir(jmeno_modulu)`
- pokud se modul změní => `reload(module)`
- vlastní moduly

# Některé další moduly

- `copy`
  - `copy(objekt1) -> objekt2`
  - `deepcopy(objekt1) -> objekt2`
- `os`
  - funkce pro operace závislé na platformě
- `sys`
  - systémové funkce a konstanty
- `time`
  - `sleep(integer)`
  - `time()`
- `re`
- `types`
- `string`
- `random`

# Modul random

## ■ pseudonáhodné generátory

- `randint(a, b)`
- `choice(seq)`
- `shuffle(x[, random])` # zamíchá sekvenci  
# na místě
- `sample(population, k)`
- `random()` #  $<0, 1$ )
- `uniform(a, b)`
- `gauss(mu, sigma)`