# Úvod do jazyka Python

# Náplň prednášok

- Málokto bude programovať čisto funkcionálne
- Veľa jazykov ale má funkcionálne prvky
- Prečo nevyužiť výhody FP
- Ukážeme si tie najčastejsie / najjednoduchšie / najzaujímavejšie …
- Príklady hlavne v Pythone

# Funkcionálny jazyk musí mať aspoň niektoré z týchto vlastností

+ **Funkcie sú objekty** (first class objects): všetko, čo viete spraviť s údajmi, viete spraviť aj s funkciami
+ **Rekurzia** sa používa ako riadiaca štruktúra
+ Zameranie na **spracovanie zoznamov**
- Čisto funkcionálne jazyky sa **vyhýbajú vedľajším účinkom**
- Čisto funkcionálne jazyky **odrádzajú od používania príkazov** (viď Výrazy a príkazy zo začiatku predmetu)
+ Používajú sa **vyššie funkcie** (higher order functions)
• Zaujímame sa o to, **čo** počítame a nie o to **ako**

– Ak už používame nejaké **premenné**, tak ich hodnota by mala byť **finálna**

Charming Python: Functional programming in Python: http://www.ibm.com/developerworks/library/l-prog/

# Python

- General purpose (jazyk hackerov)
- Multiparadigmový (PP, OOP, FP)
- Interpretovaný
- Dynamicky typovaný
- Všetko je objekt

# Zaujímavé vlastnosti

- Hlavným cieľom je pekný kód (Zen of python)
- Bohatá štandardná knižnica
- Obrovské množstvo ďalších knižníc na asi všetko
- Funkcie sú objekt

# Prečo práve python a ciele tejto časti predmetu

- Funkcionálne čtry existujú v rôznych vyšších jazykoch (JavaScript, Java, C, ...)
- Umožňujú elegantný zápis a efektívne vykonávanie
- Ukázať ako sa dá k problémom pristupovať inak a aké to prináša výhody

# Filozofia jazyku Python je programovať správne a elegantne - The Zen of Python

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one-- and preferably only one --obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea -- let's do more of those!

---

https://www.python.org/dev/peps/pep-0020/

# Na čo sa dá python použiť

- Skriptovanie (predinštalovaný v takmer každom operačnom systéme)
- Web (Django, Flask, Tornado...)
- Vedecké počítanie (NumPy, Pandas, SciPy, Matplotlib, jupyter)
- GUI (pyqt, Tk)
- Distribuované počítanie (Spark)
- Výučba (https://github.com/ipython/ipython/wiki/A-gallery-of-interesting-IPython-Notebooks)

# Nevýhody (pre tento predmet)

Toto nie je úplný zoznam:

- Nie je to funkcionálny jazyk
  - Veľmi ľahko sa dá skĺznuť k nečistým výrazom
  - Otvorené triedy
  - Chýbajú immutable objekty (dajú sa pridať)
  - Obmedzená veľkosť zásobníka (PyPy nemá)
  - Chýba optimalizácia chvostovej rekurzie
  - Pre veľa funkcionálnych čŕt chýba priama podpora
    - Väčšinou sa ale dajú ohackovať
  - Kvôli dynamickému typovaniu nemáme priamu podporu pre pattern matching
- Trochu iný objektový model ako ten, na ktorý ste zvyknutý (Keďže riešime FP a nie OOP, tak nám to môže byť skoro jedno)
  - Metódy sa nedajú priamo preťažovať
  - Nemáme kľúčové slová private, public, final ...

# Formátovanie zdrojového kódu

## PEP8

- Odsadzovanie pomocou 4 medzier a nie tab
- Prázdne riadky na oddeľovanie funkcií
- Dokumentačné komentáre
- Používanie medzery okolo znamienok
- CamelCase na ponenovanie tried a male_pismena_s_podtrznikmi na pomenovanie funkcii
- UTF-8 kódovanie zdrojových súborov
- Nepoužívajte non-ASCII znaky na pomenovávanie

---

https://www.python.org/dev/peps/pep-0008/

# Tutorial Pythonu

- Budeme používať Python 3 (konkrétne 3.4 alebo novšie)
- Spustenie interaktívneho módu

```
C:\cesta\niekam>python
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  6 2015, 01:54:25) [MSC v.1900 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Spustenie skriptu

```
C:\cesta\niekam>python script_name.py
```

# Ukážka

# Praktická ukážka

- Kalkulačka
  - + - * / ** // %
  - < > == and or
  - _
- Reťazce
  - ‘’ “” ‘’’ “”” \n
  - + * [] [-1] [:] [len(_)+vela] [:len(_)+vela]
  - konverzia: int, str
- Zoznam (list)
  - -||-
  - list() + range
  - append
  - in
- Slovník (dict)
  - {}, dict()
- print, input, len, dir, help, type (__class__)
- while, if, for

# while

```
# Fibonacciho postupnost
a, b = 0, 1
while b < 10:
    print(b)
    a, b = b, a+b
```

- Na formátovanie kódu sa nepoužívajú zátvorky
- Medzery majú význam
- **Na odsadzovanie budeme používať 4 medzery**
- Vyhodnocuje sa zprava doľava
- , sa dá použiť na viacero priradení naraz

# for

```
words = ['cat', 'window', 'defenestrate']
for w in words:
    print(w, len(w))
```

```
words = ['cat', 'window', 'defenestrate']
for i in range(len(words)): # to iste ako range(0, len(words))
    print(i, words[i])
```

```
words = ['cat', 'window', 'defenestrate']
for i, w in enumerate(words):
    print(i, w)
```

- Iteruje cez dátovú štruktúru
- Ak chcete iterovať cez čísla `range`
- Ak chcete index, `enumerate`

# if, elif, else

```
x = int(input("Please enter an integer: "))
if x < 0:
    x = 0
    print('Negative changed to zero')
elif x == 0:
    print('Zero')
elif x == 1:
    print('Single')
else:
    print('More')
```

Nič prekvapujúce

# continue, break, pass

- `continue` a `break` je to iste ako v C
- `pass` nerobí nič
  - Ak syntax vyžaduje niečo, ale vy nič nepotrebujete
  - Dobré keď len vytvárate štruktúru programu a zatiaľ nenapĺňate metódy

```
while True:
    pass  # Cakanie na prerusenie (Ctrl+C)
```

# for a while moze mat else

```
for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print n, 'equals', x, '*', n/x
            break
    else:
        # loop fell through without finding a factor
        print n, 'is a prime number'
```

# Definícia funkcie

```
def fib(n): # Fibonacciho cisla
"""Print a Fibonacci series up to n."""
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)
        a, b = b, a + b
    return(result)
```

- Kľúčové slovo `def`
- `"""` dokumentačný komentár
- `return` vracia návratovú hodnotu

# Prednastavené hodnoty

```
def ask_ok(prompt, retries=4, complaint='Yes or no, please!'):
    while True:
        ok = input(prompt)
        if ok in ('y', 'ye', 'yes'):
            return True
        if ok in ('n', 'no', 'nop', 'nope'):
            return False
        retries = retries - 1
        if retries < 0:
            raise OSError('uncooperative user')
        print(complaint)
```

- Parametre môžu mať prednastavené hodnoty
- Je možné použiť pri volaní menej parametrov
- Je možné volať názvom parametru `ask_ok('Do you really want to quit?', complaint='What?')`
- Za parametrom s prednastavenou hodnotou nemôže ísť parameter bez prednastavenej hodnoty
- Pozor!!! Prednasatvená hodnota sa vyhodnocuje v čase definície
  - Ak hodnota nie je primitývny/immutable typ, tak všetky volania funkcie budú používať rovnakú referenciu, tj. rovnaký objekt

# Premenlivý počet atribútov

```
def concat(*args, sep="/"):
    return sep.join(args)
```

```
def funkcia(argument, *arguments, **keywords):
    pass
```

- * slúži na označenie zoznamu atribútov
- Zoznam atribútov je dostupný ako `list`
- ** slúži na označenie zoznamu pomenovaných atribútov
- Pomenované atribúty sú dostupné ako asociatívne pole `dict`, kde kľúč je názov pomenovaného atribútu

# import

```
import pprint

from pprint import pprint

from pprint import *

from pprint import pprint as pp

stuff = [['spam', 'eggs', 'lumberjack', 'knights', 'ni'], 'spam',
['eggs', 'lumberjack'], 'knights', 'ni']

pp(stuff)

pp(globals())
```

```
import functools, itertools, operator
```

# Lambda výraz

```
square = lambda x: x * x
square(2)
>>> 4
```

- Velmi podobné ako funkcia
- Netreba pomenovať
- Vracia sa vykonateľný objekt
- Obmedzenia

```
def make_power(exponent):
    return lambda x: x ** exponent

square = make_power(2)
square(2)
>>> 4
cube = make_power(3)
cube(2)
>>> 8
```

# Na cviku

# Zdroje použité v prezentácii

- https://docs.python.org/3/tutorial
- https://www.python.org/dev/peps/pep-0008/
- https://www.python.org/dev/peps/pep-0020/
- https://drive.google.com/folderview?id=0ByIrJAE4KMTtaGhRcXkxNHhmY2M