NPRG065: Programming in Python

Lecture 7

http://d3s.mff.cuni.cz

Tomas Bures

Petr Hnetynka

{bures,hnetynka}@d3s.mff.cuni.cz
Functions
(continuation from lect. 5)
Type hints

- Function parameters – no explicit type defined
  - it's obvious as Python is dynamically typed
- But they can be added via type hints
  - since python 3.5
  - only for documentation purposes!
  - still no type checking at runtime!

```python
def greeting(name: str) -> str:
    return 'Hello ' + name
```

See hints.py
Anonymous functions

- `adder = lambda x, y: x + y`
- `print_val = lambda name, value: name + '=' + str(value)`

Lambda body ~ single expression

- rather limited
  - Python authors do not like lambdas
    - but it is not a big deal; regular functions are first class objects, references to them can be passed
Functional programming (FP)
- declarative programming paradigm
- computation as the evaluation of mathematical functions
- avoids changing-state and mutable data

Python builtin functions for FP
- `map` and `filter`
- `enumerate`, `sorted`, `any`, `all`, `zip`
- module `functools`
- and `operator`
Generators

- When you need elements of a sequence but not the complete sequence
  - similar to an iterator
- Generator functions
  - a function with yield instead of return
  - yield – allows functions to suspend and resume their state between each call

```python
def get_squares_gen(n):
    for x in range(n):
        yield x ** 2
```

- Generator expressions
  - similar to list comprehensions, but
  - return an object that produces results one by one
    - instead of directly producing a list

```python
(k**2 for k in range(10))
```
Back to core types
• Supports “big-size” integers
• Internal representation
  ▪ till \texttt{sys.maxsize} – regular int
  ▪ over \texttt{sys.maxsize} – a sequence of digits

\begin{verbatim}
import sys
import math
math.log(sys.maxsize, 2)
# prints out size of "small" integers in bits
\end{verbatim}

• \texttt{int} is a class
  ▪ integers are objects (instances of the \texttt{int} class)
    • classes will start next lecture
  ▪ is not computing inefficient? (i.e., creating too many objects)
  ▪ a pool for the commonly used numbers (-5 to 256)
floats are inherently imprecise
- internally represented as base 2 fractions
  - “human floats” are base 10 fractions

```python
print(0.1 + 0.1 + 0.1 == 0.3) # -> False
print(1/10 + 1/10 + 1/10 == 3/10) # -> False
```

Decimal and Fraction types
- exact representation
  - but slower computations

See nums.py