Programming in Python

Syntax and basic constructs
lecture 1

Department of Cybernetics and Artificial Intelligence
Technical University of Košice
Ing. Ján Magyar, PhD.

History of Python

- stemming from the 80s - Guido van Rossum
- 1990 - Python 1.0
- 2000 - Python 2.0
- 2008 - Python 3.0
- currently - Python 3.11.X

Python characteristics

- general
- higher
- interpreted
- multi-paradigm
  - **structural/procedural programming**
  - **object-oriented programming**
  - functional programming
  - partially aspect-oriented programming
  - partially metaprogramming
  - logical programming through extensions

Python vs. C

● interpreted	● compiled
● higher-level language	● mid-level language
● multi-paradigm	● procedural programming
● dynamic type check	● static type check
● variables, garbage collection	● references, memory allocation
● support for defining data structures	● explicit data structure definition
● syntactically significant indentation	● code blocks with brackets

Basic constructs - overview

- values and variables
- operators
- branching
- loops
- functions

Values and variables

- dynamic type check
- defining a variable in C:

```
type name = value;
int number = 5;
```

- defining a variable in Python:

```
name = value
number = 5
```

Naming variables

„There are only two hard things in Computer Science: cache invalidation and naming things.“

-- Phil Karlton

- keywords are restricted
- do not use names of standard methods and functions
- do not use letters `O`, `I`, `l`
- names start with letters
- short and easy-to-understand

Naming conventions

- variables, methods and functions: lowercase, words separated by _
  `my_wonderful_variable`, `my_wonderful_function`
- classes: capital letters, camelcase
  `MyClass`
- constants: uppercase, words separated by _
  `MY_CONSTANT`
- modules: lowercase, words separated by _
  `my_module`
- packages: lowercase, words not separated
  `mypackage`

Primitive types in Python

- integer
- float
- complex (e.g. `3 + 4j`)
- boolean (`True`, `False`)
- string (e.g. `'abc'` or `"abc"`)
- `None`

Sequence types in Python

- `list`
  - mutable
  - sequence of values of different types (usually homogeneous)
  - `[1, 2.4, 'abc']`
- `tuple`
  - immutable
  - sequence of values of different types (usually heterogeneous)
  - `(1, 2.4, 'abc')`
- `range`
  - immutable
  - consists of integers
  - three parameters: `start`, `stop`, `step`
  - `range(3, 8)`

Mapping types in Python

`dictionary`

- maps hashable values to arbitrary values
- consists of key-value pairs
- you cannot use as a key: list, dictionary, mutable values
- `dct = {'boys': ['', '', ''], 'girls': ['', '']}`

Set types in Python

- unordered set of unique hashable values
- used for:
    - membership tests
    - removing duplicates
    - set operations
- `set`
    - mutable
    - unhashable
    - `{'ab', 'bc'}`
- `frozenset`
    - immutable
    - hashable

Operators in Python

- arithmetic operators
- assignment operators
- comparison operators
- logical operators
- identity operators
- membership operators

Arithmetic operators in Python

+	addition
-	subtraction
*	multiplication
/	division
%	modulo (remainder)
//	integer division
**	exponentiation

Assignment operators in Python

=	x = 5	&=	x = x & 5
+=	x = x + 5	|=	x = x | 5
-=	x = x - 5	^=	x = x ^ 5
*=	x = x * 5	>>=	x = x >> 5
/=	x = x / 5	<<=	x = x << 5
%=	x = x % 5		
//=	x = x // 5		
**=	x = x ** 5		

Walrus operator

```
(x:=5)
```

- from Python 3.8
- assigns a value and returns it
- criticized
  - only one operator should exist for each operation
  - simplicity is better than complexity
  - nobody knows how it will be used

Using the walrus

```
# f() may return None
x = f()
if x:
    process(x)
```

```
[f(x), f(x)**2, f(x)**3]
```

```
# f() may return None
if (x:=f()):
    process(x)
```

```
[y:=f(x), y**2, y**3]
```

Comparison operators in Python

==	equals
!=	does not equal
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

Logical operators in Python

`and`	all are true
`or`	at least one is true
`not`	negation

Identity operators in Python

```
is

is not
```

Membership operators in Python

`in`	exists
`not in`	does not exist

Branching – conditional statement

```
if condition:
    body
elif condition:
    body
else:
    body
```

Loops

generally three types:

1. arithmetic
    - for
2. logical
    - **while**
    - do … while
3. **foreach**

Logical loops in Python

```
while condition:
    body
```

do … while in Python

```
body
while condition:
    body
```

Foreach loop in Python

- iterating over the members of a sequence

```
for e in sequence:
  (do something with e)
```

- the sequence can be:
  - list
  - tuple
  - `range`
  - `set/frozenset`
  - string – members are letters

Arithmetic loop in Python

- C-based languages offer `for`:

```
    for (int i = 0; i < 5; i++) { body; }
```

- representation in Python

```
    for i in range(0, 5):
        body
```

- setting the iterator update is possible by using `step`

Conclusion

- basic Python characteristics
- taxonomoy of basic constructs
- taxonomy of operators
- branching and loops