Python

#2

Agenda

- Loops in Python
    - For loop
        - iterating over a sequence
        - iterating specific number of times
    - While loop
        - Stop/continue conditions
    - Comparison and logical operators
    - If instruction
- Conditional instruction if...elif...else
- Basic string manipulation

Who am I?

Michał Gałka

Contact:

galka.michal@gmail.com (e-mail or hangouts)

Use **[BioIT]** in the e-mail topic.

Who am I?

Michał Sarna

Contact:

michalsarna@gmail.com (e-mail or hangouts)

Use **[BioIT]** in the e-mail topic.

Who am I?

Marcin Górski

Contact:

mkgorski@gmail.com (e-mail or hangouts)

Use **[BioIT]** in the e-mail topic.

Who am I?

Andrzej Stasiak

Contact:

aandrzej.stasiak@gmail.com (e-mail or hangouts)

Use **[BioIT]** in the e-mail topic.

Loops and conditions

For Loop

- Iterates over a sequence.
- Loop control variable gets subsequent values from the sequence.
- All instructions in the block after the **`for`** instruction are executed for each member of the iterable.

For loop

```python
items = ['a', 'b', 1, 'c', 'd', 2, 'e', 'f', False]
for i in items:
    print(i)
```

```
a
b
1
c
d
2
e
f
False
```

For loop

```
items = ['a', 'b', 1, 'c', 'd', 2, 'e', 'f', False]
for index, value in enumerate(items):
    print(index, value)
```

```
0 a
1 b
2 1
3 c
4 d
5 2
6 e
7 f
8 False
```

For loop

```
for x in range(10):
    print(x)
```

```
0
1
2
3
4
5
6
7
8
9
```

Comparison operators

- There are several comparison operators in Python
- They perform a certain comparison type on operands provided.
- The result of a comparison is always either **`True`** or **`False`**.

Comparison operators

- `==` - equal
- `<` - less than
- `<=` - less or equal
- `>` - greater than
- `>=` - greater or equal
- `!=` - not equal

Logical operators

- Logical operators perform boolean operations on the operands.
- The result of logical operators is always either **`True`** or **`False`**.

Logical operators

- **`and`** - True if operands on both sides return True
- **`or`** - True if at least one of the operands returns logical True
- **`not`** - unary operator, that changes operands logical value to the opposite one.

while loop

- File loop executes an instruction block while the condition specified remains True.
- It’s possible to not run at all (if the condition is False at first check).
- It’s possible to run forever (if the condition never becomes False).

While loop

```python
x = 10
while x > 0:
    print(x)
    x = x - 1
```

```
10
9
8
7
6
5
4
3
2
1
```

break and continue

- There are 2 statements in Python that can be used in both **`while`** and for loops.
- If interpreter meets **`break`** statement in the loop block it leaves the loop and continues execution of the rest of the program.
- If interpreter meets **`continue`** statement it ends the current loop iteration immediately and continues with the subsequent one.

Conditional instruction

- There is only one conditional instruction in Python: **`if...elif...else`**.
- All instructions in the block after the **`if`** statement are executed only if the condition is True.
- There can be any number of **`elif`** blocks.
- The elif condition is checked only if all previous **`if`** and **`elif`** conditions were False.
- The block after else statement is executed only if none of the previous **`if`** and **`elif`** blocks was True.
- **`elif`** and **`else`** blocks are optional.

if

```
x = -10
if x != 0:
    print('x is not zero')
```

if...elif

```python
if x < 0:
    print('x is a negative number')
else:
    print('x is not a negative number')
```

if...elif...else

```
if x < 0:
     print('x is a negative number')
elif x == 0:
     print('x is zero')
else:
    print('x is a positive number')
```

String manipulation

Concatenation

- The easiest way to concatenate strings in python is to use + operator
- However you need to remember that it is not the fastest and the most memory efficient method.

```
>>> s = 'Brave ' + 'Sir ' + 'Robin '
>>> s
'Brave Sir Robin '
>>> s = 'Sir Robin'
>>> s
'Sir Robin'
>>> s = 'Brave ' + s
>>> s
'Brave Sir Robin'
```

String formatting

- Each string has a possibility to put arbitrary data in it.
- Data are inserted into the places marked with **{}**
- This method is much more efficient than concatenation with **+**

String formatting

```
>>> 'Sir {} the {}'.format('Robin', 'Brave')
'Sir Robin the Brave'
>>> name = 'Robin'
>>> nickname = 'Brave'
>>> full_name = 'Sir {} the {}'.format(name, nickname)
>>> print(full_name)
Sir Robin the Brave
```

split and join

- `split` method of a string divides it into a list of chunks.
  - Chunks are delimited by a separator provided as a split argument.
- `join` method of a string takes a list of chunks and produces a string.
  - Chunks are separated by a string for which the method was called.

split and join

```
>>> s = 'Brave Sir Robin'
>>> s.split(' ')
['Brave', 'Sir', 'Robin']
>>> s.split()
['Brave', 'Sir', 'Robin']
>>> l = s.split()
>>> l
['Brave', 'Sir', 'Robin']
>>> '-'.join(l)
'Brave-Sir-Robin'
```

startswith and endswith

- **`startswith`** returns **`True`** if the string starts with specific characters provided as an argument.
- **`endswith`** returns **`True`** if the string ends with specific characters provided as an argument.

startswith and endswith

```
>>> s = 'He bravely turned his tail and fled.'
>>> s.startswith('He')
True
>>> s.endswith('fled')
False
>>> s.endswith('d.')
True
>>> s.endswith('fled.')
True
```

find and rfind

- **`find`** returns the lowest index of the string where the substring occurs.
- **`rfind`** returns the highest index of the string where the substring occurs.

find and rfind

```
>>> s = 'Repeated Acronym Syndrome Syndrome'
>>> s.find('Repeat')
0
>>> s.find('Syndrome')
17
>>> s.rfind('Syndrome')
26
>>> s.find('unknown')
-1
```

strip, lstrip and rstrip

- `strip` method removes leading and trailing whitespaces if it’s run without any arguments.
  - It can remove arbitrary characters when provided as an argument.
- `lstrip` removes only leading characters.
- `rstrip` removes only trailing characters.

strip, lstrip and rstrip

```
>>> '   Brave Sir Robin   '.strip()
'Brave Sir Robin'
>>> '   Brave Sir Robin   '.lstrip()
'Brave Sir Robin   '
>>> '   Brave Sir Robin   '.rstrip()
'   Brave Sir Robin'
>>> s = '  Brave Sir Robin   '
>>> s.strip()
'Brave Sir Robin'
```