Introduction to Programming in Python

Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Methods

Apart from functions (like `len` and `print`), Python also has *methods*. They work almost like functions, but they are called by appending the name of the method to the name of an object (here, a string):

In [ ]:
```
s1 = "Simon"
```

In [ ]:
```
s1.upper()
```

Objects of different types support different operations (methods). Here is a list for strings:

In [ ]:
```
print(', '.join(filter(lambda m: callable(getattr(s1, m)) and not m.startswith("_"), dir(s1))))
```

The key in the command above is `dir(s1)` (try it). The rest is just for pretty printing.

Another way to find out which methods are supported by an object of a given type is to use *autocompletion*:

In [ ]:
```
# uncomment the next line
# s1. # try hitting the tab key after the dot
```

You can find out what a method does by using the `help` facility:

In [ ]:
```
help(s1.upper)
```

Exercise

The following methods for strings are needed in the homeworks. Try to find out what they do, using a mix of `help` and try and error: `lower`, `upper`, `capitalize`, `startswith`, `endswith`, `index`, and `find`.

In [ ]:

Lists

Lists are indexable collections of arbitrary (though usually homogeneous) things:

In [ ]:
```
list1 = [1, 2., 'hi']; print(list1)
```

As for strings, the function `len` returns the length of a list (or any other sequence):

In [ ]:
```
len(list1)
```

Like strings, they support indexing, but unlike strings, they are *mutable*, i.e., elements of the list can be changed

In [ ]:
```
list1 = [1, 2., 'hi']
list1[2] = 42
print(list1)
```

The `sum`, `min`, and `max` functions respectively compute the sum, minumum, and maximum of a list, provided this is meaningful considering the elements of the list:

In [ ]:

```
list1 = [1, 2., 42]
print(sum(list1))
print(min(list1))
print(max(list1))
```

Like strings, lists support a number of methods:

In [ ]:
```
print(', '.join(filter(lambda m: callable(getattr(list1, m)) and not m.startswith("_"), dir(list1
```

Exercise

The `append`, `insert`, `index`, `remove`, `reverse`, and `count` methods are needed for the homeworks. Find out what they do.

In [ ]:
```

```

Tuples

- A `tuple` is similar to a list, but *immutable*. It is created with round brackets:

In [ ]:
```
(1, 2., 'hi')
```

Exercise

1. Create a list containing the names "Simon", "Carl", and "Lucy" as strings, and store it in a variable.
2. Change the second element of the list to "Karl".
3. Repeat, but now using a tuple instead of a list. Why does this fail?

In [ ]:

Control Flow

Conditionals

Conditional statements are used if code is only to be executed if some condition holds.
Example:

In [ ]:
```
x = int(input("Enter a positive number: "))
if x < 0:
    print("You have entered a negative number.")
```

Notes:

1. Code blocks are introduced by colons and have to be indented.
2. The `if` block is executed if and only if the condition is `True`.

What if we want to do something in case the condition is `False`, as well? `else` to the rescue.

In [ ]:

```
x = int(input("Enter a positive number: "))
if x < 0:
    print("You have entered a negative number.")
else:
    print("Thank you.")
```

Evidently, the `else` block is executed whenever the condition is `False`.

What if we have multiple conditions? E.g., we want a number between 0 and 9? Solution: `elif` (read: else if).

In [ ]:

```
x = int(input("Enter a number between 0 and 9: "))
if x < 0:
    print("You have entered a negative number.")
elif x > 9:
    print("Your number is greater than 9.")
else:
    print("Thank you.")
```

Notes:

1. The `elif` block is executed when the `if` condition is false, but the `elif` condition is true.
2. Exactly one of the three blocks is executed. The `else` block acts as a catch-all if none of the others is triggered.
3. There could be more than one `elif` block, but only one `if` and only one `else`.

Exercise

The body mass index (BMI) is defined as a person's weight (in kg) divided by the squared height in meters. The following thresholds exist (source: NHS):

- below 18.5 – underweight
- 18.5 up to 25 - healthy
- 25 up to 30 - overweight
- above 30 - obese.

Write a program that asks the user for their weight and height, calculates the BMI, and prints a message telling the user `You are ... .`

Other built-in datatypes

- Other built-in datatypes include `set`s (unordered collections) and `dict`s (collections of key-value pairs). More on these later.

Homework

- Lists, methods: Beginner exercises 6-7, 9-10, 17, 18 from https://holypython.com/
- Conditionals:
    - Intermediate exercises 7a, 7b from https://holypython.com/
    - Exercises 2, 31, 33, and 40 from https://www.w3resource.com/python-exercises/python-conditional-statements-and-loop-exercises.php

Note for the sample solution of Ex. 33: you can test if a list or tuple contains a given value like this:

In [ ]:
```
3 in [1, 2, 3]
```