FUNCTIONS AND CLASSES IN PYTHON

Aims of the Lecture

- Learn and understand the purpose of using functions and classes in Python.
- Implement your own functions.
- Learn how classes work by understanding the basic notions of object-oriented programming (OOP).

Additional Reading and Sources

- Real Python (https://realpython.com/lessons/classes-python/)

Functions()

- One of the key components of mathematics and computing code.

- $x \rightarrow y$: given an input x, you obtain an output y.
  - The classical example is the straigh line function, $y = mx + b$

- In computing, functions also take inputs and outputs.

- In fact, we have already seen several pre-built functions in Python during previous lessons!
  - Can you mention any of them?

In [ ]:
```
# The print function taking a string as an input.
# The output is what you see printed.
print('Hello World!')
```

- Python has a long list of pre-built functions that are located somewhere in the installation files.
  - Don't worry, we don't care where they are!

- These functions have been built by someone, and therefore we should also be able to build our own.

- Python (and most programming languages) give us the opportunity to create our own functions.

How to create a Python funtion?

1. First think about what you need the function for.
    A. Google it!
    B. Consider which are the inputs and the outputs that you require.

1. Name your function with something you remember or relates to what the function does.

1. **At the beginning of your code**, *define* the function by using *def*, then the name of the function and a parenthesis where you will indicate the inputs. Finalise the line with ":"

1. Once you press enter, there will be an indent. Start by inserting three single quotes (six in total when the after the IDE autocompletes) and write the description of your function.

1. Press enter and leave a space for the actual content.

1. Finally, use *return* to indicate that the function is finished. After the word return, you will indicate your outputs.

In [ ]:

```
# The skeleton of your Python function

def myfunction(i1, i2, i3):
    '''This function does whatever I want'''

    return o1, o2, o3
```

Why creating a function?

- Sometimes you need to call the same code more than once.

In [ ]:
```
## This code uses a foor loop where numbers are added to a variable called "acc".

numbers = list(range(10))
print('The list of numbers is,',  numbers)
acc = 0
for n in numbers:
    acc = acc + n
    print('The accumulated value is', acc)
```

This is what we call an **accumulator**.

What would happen if we receive many lists of numbers at different parts of our program and we want to accumulate them?

Creating and using a function

Create & execute *accumulate()* ("nothing" will happen).

In [ ]:
```
def accumulate(numbers, acc=0):
    '''This function takes a list and an accumulator value as an input and delivers
the accumulated values as output.'''
    print('The list of numbers is,',numbers)
    for n in numbers:
        acc = acc + n
        print('The accumulated value is', acc)
    return acc
```

- Two inputs:
  - *numbers* is a list of numbers.
  - *acc* is the current value of the accumulator.

- *acc* equal to zero? Why?

- One output, *acc* (same as the input!), is this correct?

Example of using the *accumulate()* function:

In [ ]:
```
a = [1,2,3]
accumulate(a)
```

- Notice that we didn't had to indicate an initial accumulator *acc* value, and therefore our function assumed it was zero.

- Also notice that the list of numbers that we input to our function can have any name we want!
  - As long as the inputs are in the right position, the function should work!

Storing the output

- We only applied the function, but we didn't store the output!

In [ ]:
```
a = [1,2,3]
b = accumulate(a)
print(b)
```

- We can use a new list of numbers and the accumulator b to keep accumulating more values from other lists.

In [ ]:
```
a = [2,1,4,5,0,67]
b = accumulate(a,b)
```

What happens if we run again the cell above?

Classes

- Just as there are built in functions and we need to create our own, sometimes we need to create our custom data types!

- This is the main reason to use classes.

- Object-Oriented Programming (OOP): Programming paradigm where objects are manipulated to obtain results. Wikipedia (https://en.wikipedia.org/wiki/Object-oriented_programming)

Example: People as Objects

- A data type is a noun, for example a person

- A person has certain properties such as
  - Name
  - Age
  - Address

- A person has certain behaviours such as
  - Walk
  - Talk
  - Breathe

- If we think of a **class** that creates persons (objects), every person will have a name, age and address particular to that individual.

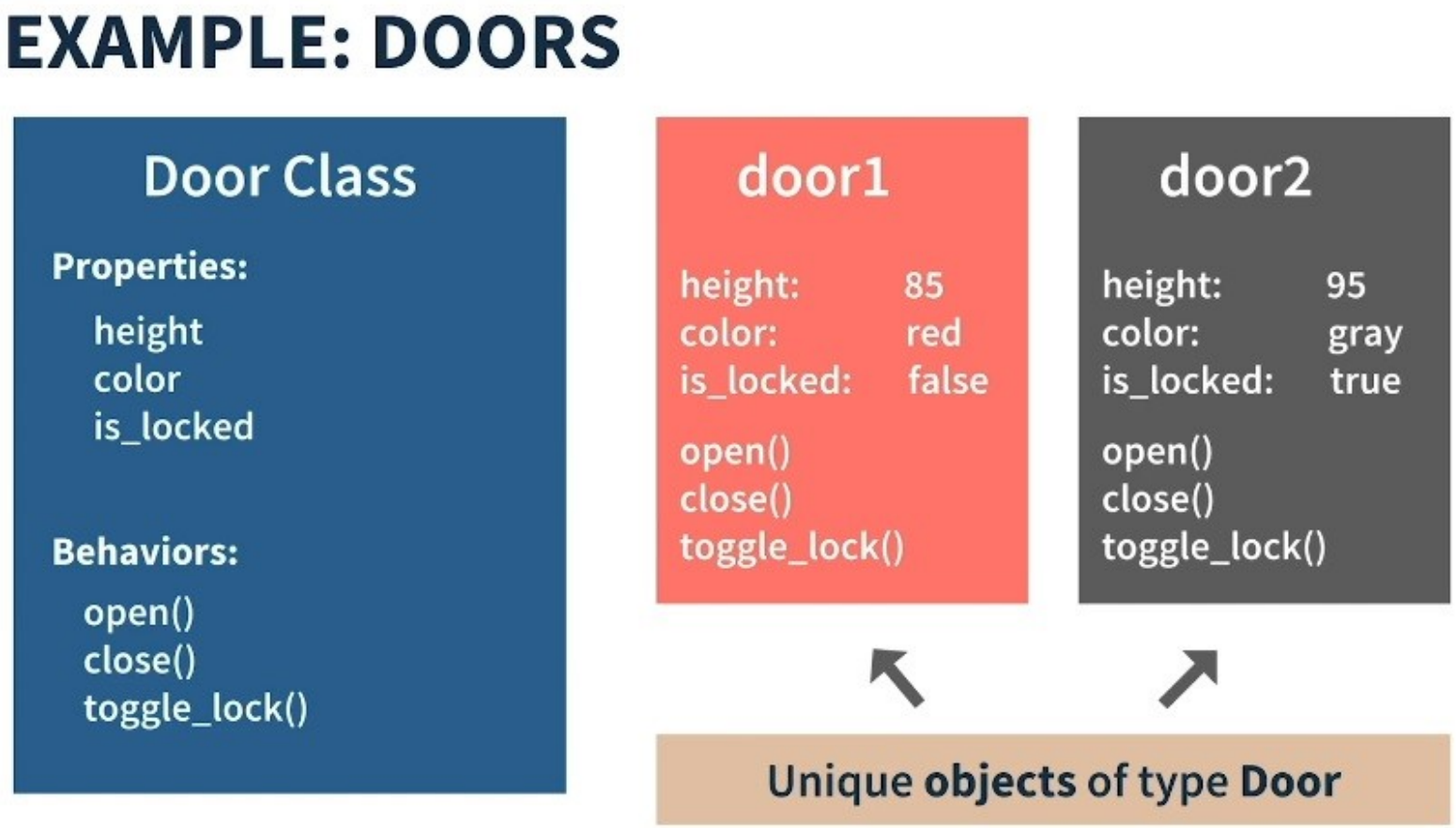
- Moreover, all of those individuals will be capable of walking, talking and breathing.

- When we create an object from a certain class, this is called **instantiation**

In [ ]:
```
## instantiation of a string
a = 'Hello'
```

- We are unable to see it, but there is a class that creates (instantiates) the string "Hello" into the variable "a".

Example 2: A program that creates doors



Classes in Python

Creating a class

- First, we define a class by using the *class* operator:

In [ ]:
```
class Person:
    pass
```

- Classes contain *attributes*. There are two types:

- **Instance Attributes**: Unique to each object. Any person we create will store its name and age. We can change the attributes without affecting other objects

In [ ]:
```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

- The **init** function is called an **initializer**.

- It is automatically called when we instantiate the class.

- It is used to make sure that the class has the default attributes that are needed for the object.

- Also used to verify that attributes are entered correctly (i.e. the age is not negative, the name is a string, etc.)

- **Class Attributes**: Unique to each class. Used to specify a default value for all objects.

In [ ]:
```
class Person:

    species = 'mammal'

    def __init__(self, name, age):
        self.name = name
        self.age = age
```

Creating an object

- Let's instantiate the Person class to create John, age 18:

In [ ]:
```
p1 = Person("John", 18)
print(p1)
```

- We can check the attributes (instance & class) of person *p1* by using the dot notation:

In [ ]:
```
p1.name
```

In [ ]:
```
p1.age
```

In [ ]:
```
p1.species
```

Adding methods to the class

- To add a behaviour to our person, we use methods. With the following code we are adding a *talk* method to the *Person* class:

In [ ]:
```
class Person:

    species = 'mammal'

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def talk(self, speech):
        print(self.name+' says: '+speech)
```

In [ ]:
```
p1 = Person("John", 18)
p1.talk("Hello everyone!")
```

QUESTION: Do you think everything that can be done with functions is possible with classes and/or vice versa?