Industrial Software Development

Tutoring sessions

Tutor: Maura Pintor (maura.pintor@unica.it)

Exercises

1. Write a function that takes every element in a list and print its type. (5 minutes)
2. Write a function that creates one dictionary given a list of keys and a list of values. (5 minutes)
3. Write a function that finds the minimum value of each list contained in a dictionary. Write it with and without using the `min` function. (10 minutes)
4. Write the `guess_number` function that accepts an input from the user and gives suggestions until the secret number is guessed. (10 minutes)

Advanced Python

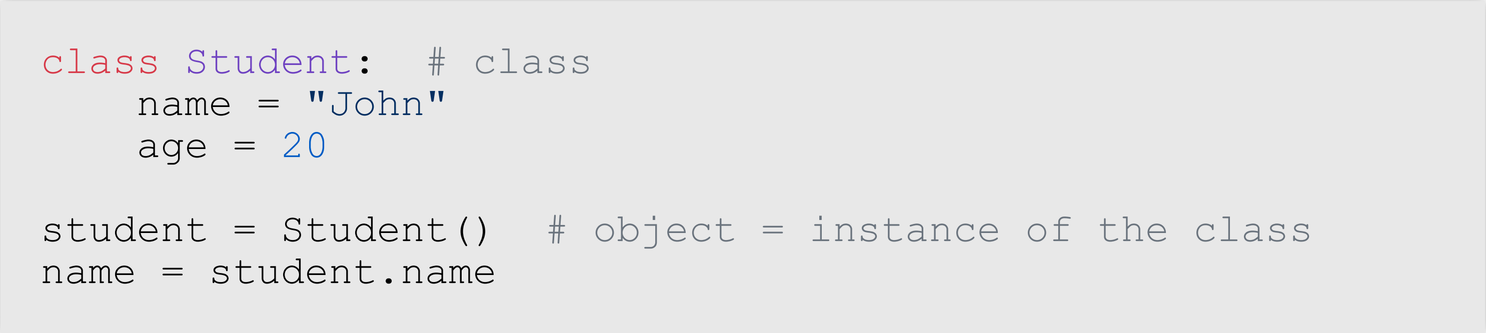
Python Classes / Objects

Objects are complex Python structures, with their properties and methods.

- property: variables stored in the object
- method: functions that are accessible from the object

Classes are object creators, they define the blueprint of the object.

Python Classes / Objects



```python
class Student:  # class
    name = "John"
    age = 20

student = Student()  # object = instance of the class
name = student.name
```

The `__init__` function

All classes have a function called `__init__()`, which is always executed when the class is being initiated.
We can use the `__init__()` function to assign values to object properties, or other operations that are necessary to do when the object is being created.

The `__init__` function

Within the class, we can refer to `self` to get (or assign) attributes and methods stored in the class.

The `__init__` function

```
class Student:  # class
    def __init__(self, name, age):  # self is passed as
input
        self.name = name  # store attributes in self
        self.age = age

student = Student('John', 20)  # now we can pass values
here
name = student.name
```

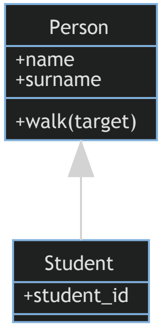
The `__init__` function

Now we can define methods and access attributes stored in self.

```
class Student:  # class
    def __init__(self, name, age):  # self is passed as
input
        self.name = name  # store attributes in self
        self.age = age

    def print_info(self):
        # access the attributes of the object
        print(f"Student {self.name} is {self.age} years
old.")

student = Student('John', 20)  # now we can pass values
here
student.print_info()
```

Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class.



Inheritance

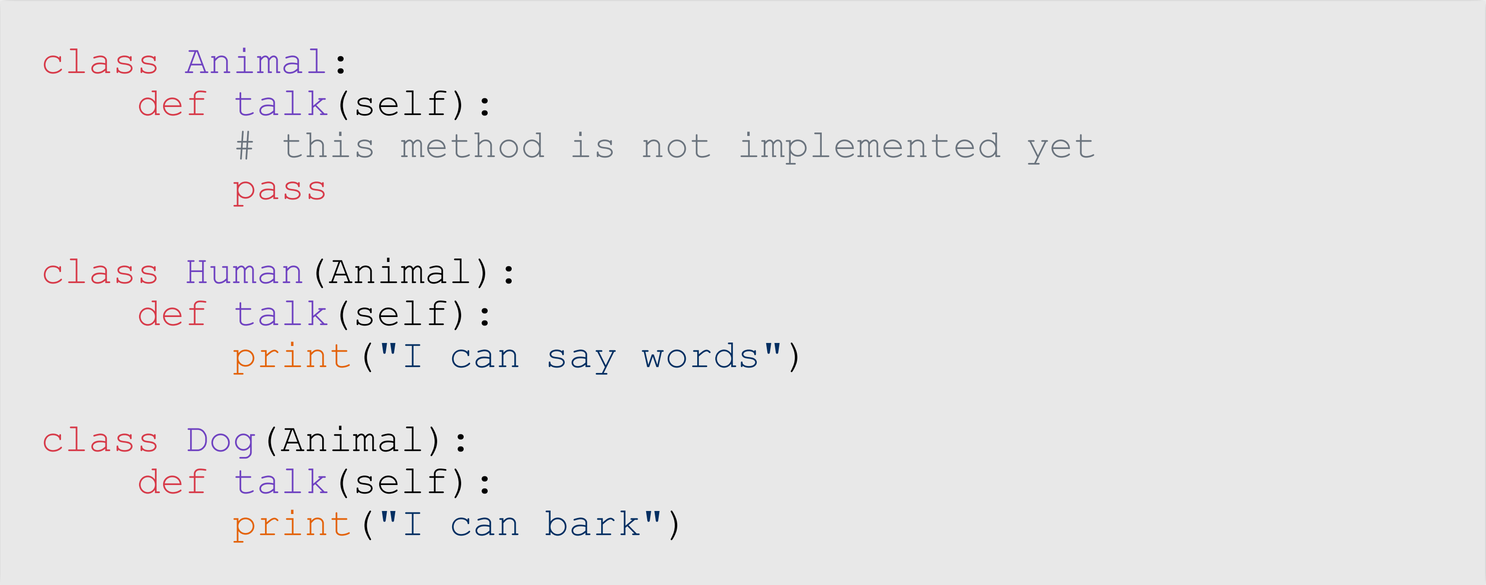
```python
class Person:
    def __init__(self, name, surname):
        self.name = name
        self.surname = surname

    def walk(self, target):
        print(f"{self.name} {self.surname} is walking to
{target}")
```

Inheritance

```
class Student(Person):
    def __init__(self, name, surname, student_id):
        self.student_id = student_id
        super().__init__(name, surname)  # calls method
from parent

    def print_info(self):
        print(f"Student: {self.name} {self.surname} ID:
{self.student_id}")

st = Student("John", "Doe", "12345")

st.walk("the store")  # inherits parent method
st.print_info()
```

Abstract objects and methods

We can define a class that is not supposed to be ever instantiated, but defines a blueprint for creating other classes.
The other classes cannot be instantiated if they don't implement *at least* this method.



```python
class Animal:
    def talk(self):
        # this method is not implemented yet
        pass

class Human(Animal):
    def talk(self):
        print("I can say words")

class Dog(Animal):
    def talk(self):
        print("I can bark")
```

Abstract objects and methods

Sometimes you will find also the `...` instead of `pass`.

```
class Animal:
    def talk(self):
        # this method is not implemented yet
        ...
```

Checking if an object is an instance of a class

```
h = Human()
print(isinstance(h, Human))
print(isinstance(h, Dog))
print(isinstance(h, Animal))
```

Exercises with classes (20 minutes)

1. Write a Python class `Vehicle` with instance attributes `max_speed` and `current_speed`, that are passed when creating the objects. Set the default value of `max_speed` to 200.
2. Implement the method `set_speed` of the class, that takes as input the desired speed and sets `current_speed` equal to it. Remember to check if the target speed is greater than `max_speed`, and if it is, print a message to the user and set the speed to `max_speed` instead.
3. Create a Python class `Bus` that inherit the `Vehicle` class and has the additional attribute `max_capacity` and `occupied_seats`.

Exercises with Python classes

We can use now classes from external libraries. Let's use the `turtle` module.

```
import turtle

t = turtle.Turtle()   # this is a python object
t.forward(100)   # calls a method from this class
```

Turtle module

We can make it more user-friendly by using another class that works together with the Turtle class. Don't worry about this part now, it is sufficient to add a few lines at the beginning and end of our script.

```
import turtle

wn = turtle.Screen()  # class Screen

t = turtle.Turtle()  # class Turtle

t.forward(100)  # method of class Turtle
wn.exitonclick()  # method of class Screen
```

Turtle module

```
t.shape("turtle")  # changes the shape of the turtle
t.color("green")  # changes the shape of the turtle
```

Turtle module

Let's draw a square with our turtle!

```
t.forward(100)
t.left(90)
t.forward(100)
t.left(90)
t.forward(100)
t.left(90)
t.forward(100)
```

Now implement it with a `for` loop.

Turtle module

Exercise: implement a function that takes as input a turtle instance and the number of sides and draws a polygon. Remember that the sum of the internal angles of a regular polygon is `(180*(sides - 2))`.

Polygon with the Turtle module

```
def draw_polygon(t, sides=3):
    if sides < 3:
        print("Cannot draw polygon with less than three
sizes!")
    angles = (180 * (sides - 2)) / sides
    for _ in range(sides):
        t.forward(100)
        t.right(180 - angles)
```

Exercise

Implement a class called `MyTurtle` that creates a red turtle instead of the default turtle. Use inheritance to keep all the existing functionalities of the `Turtle` class. Pass it to the function just implemented and check that it still works, but with our custom turtle.

Raise errors

What if a user asks for a polygon with less than three sides? We can raise errors in our code. They are nicer than print statements because the block the execution and return the error to the user.

```
def draw_polygon(t, sides=3):
    if sides < 3:
        raise ValueError("Cannot draw polygon with less
than three sizes!")
    angles = (180 * (sides - 2)) / sides
    for _ in range(sides):
        t.forward(100)
        t.right(180 - angles)
```

Try-except constructs

If we don't know if the error might happen or not, we can use the try-except construct.

```
try:
    draw_polygon(t, sides=-1)
except ValueError:  # specific except
    draw_poligon(t, sides=5)
except:  # vague except
    draw_polygon(t, sides=10)
```