## → Python User Input

Python allows for user input, using the input() method

```
username = input ("Enter username:")
print ("Username is:" + username)
```

## → Python OOP

### 1. Classes / Objects

. Python is an object oriented programming language
. Almost everything in Python is an object, with its properties and methods
. A class is like an object constructor, or a "blueprint" for creating objects

#### ↳ Create a Class

To create a class, we use the keyword class:

```
class MyClass:
```

## The __init__() function

All classes have a function called __init__() which is always executed when the class is being initiated.

We use the __init__() function to assign values to object properties.

It's like the constructor in Java

Example:

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

(reference to the instance of the class; attributes)

## The __str__() function

The __str__ function controls what should be returned when the class object is represented as a string

```
    def __str__(self):
        return f"{self.name} ({self.age})"
```

## Creating objects

Once a class is defined, we can create objects (instances of the class)

```
p1 = Person("Nour", 15)
print(p1) # output Nour (15)
```

↳ Attributes and Methods

. Attributes are data members, or properties about objects.

. We can access them using the dot '.' notation without the need to write a get() function like in Java.

Ex:

```
print (p1.name) # accessing the attribute 'name'
# output : Nour
```

. we can also modify the property or the attribute value using the '=' notation without the need to creat a set() function like in Java.

Ex:

```
p1.age = 11
print (p1). # output: Nour(15)
```

. Classes can also contain methods. Methods are functions that can be applied on objects

Example:

```
# insert a function into the class Person
def sayHello (self)
    print("Hello my name is" + self.name)
# Call the function on the person 'p1'
p1.sayHello() # output: Hello my name is Nour
```

→ **Class Variable**

Class Variables are variables shared among all instances of a class. They are defined outside any method in the class.

Example:

```
class Employee:
    # class variables
    company = "ABC Corporation"
    nbOfEmployees = 0

    def __init__(self, name, position)
        # instance variables
        self.name = name
        self.position = position
        Employee.nbOfEmployees ++

    def __str__(self)
        return f"{self.name} works at {self.company}
                as a {self.position}."

# creating instance of the Employee class
c1 = Employee("John Doe", "Software Engineer")
c2 = Employee("Jane Smith", "Project Manager")

# Accessing instance variables
print(c1)  # Output: John Doe works at ABC corporation as
                    a Software Engineer
print(c2)  # Output: Jane Smith works at ABC corporation
                    as a Project manager
```

```
# Accessing class Variable:
print (f "There are {Employee.nbOfEmployees} empl
        at {Employee.company}.")
# Output: There are 2 Employees at ABC corporation
```

- Modifying a class variable affects all instances
- If an instance has an instance variable with the same name of a class variable, the instance variable will overshadow the class variable for that instance

```
Class Employee
    company = "ABC corporation"

    def __init__(self, name):
    # Instance variable with the same name as class variable
    self.company = "DEF group"

e1 = Employee("Jhon")

print(f"{e1.name} work at {e1.company}")
# Output: Jhon work at DEF group

print(f"{e1.name} work at {Employee.company}")
# Output: Jhon work at ABC corporation
```

## Python Inheritance

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

Parent class is the class being inherited from, also called base class

Child class is the class that inherits from another class, also called derived class

### Create a Parent Class

Any class can be a Parent class

```
Class Person:
    def __init__(self, fname, lname):
        ...
    def __str__(self):
        ...
```

### Creat a Child Class

To create a class that inherits the functionality from another class, we send the parent class as a parameter when creating the child class

```
Class Student(Person):
    pass  (!! use "pas" when we don't want to add anything to the class)
```

Now the Student has the same properties and methods as the Person class

↳ Add the __init__() Function

Add the __init__() function to the Student class:

```
Class Student (Person):
  def __init__ (self, fname, lname)
    super().__init__ (fname, lname)
```

Now we have successfully added the __init__() function and kept the inheritance of the parent class.

↳ Add Properties

We can add in the __init__() function the child class specific properties. in addition of properties inherited from the father class:

```
Class Student (Person):
  def __init__(self, fname, lname, year):
    super().__init__ (fname, lname)
    self.graduationYear = year

x = Student ("Mike", "Olsen", 2025)
```

↳ Add Methods

. We can also add to the child class its own methods

. If we add a method in the child class with the same name as a method in the parent class, the inheritance of the parent class will be overriden