HSLU Lucerne University of Applied Sciences and Arts

Python for Data Science: SW07

OOP part I

Information Technology

April 2, 2025

FH Zentralschweiz

Content

- Object-oriented programming
  - What is it?
  - Classes and Objects
  - Class Components (init, self, methods, class variables)
- Class Syntax
  - Class Header and Documentation
  - Class Variables
  - Constructor
  - Object Variables
  - Methods
- Print an object
  - `__str__()` method

HSLU Lucerne University of Applied Sciences and Arts

Object Orientated Programming: What is it?

Information Technology

April 2, 2025

Object-Orientated Programming: What is it?

In contrast to functional programming, the concept of object-orientated programming provides an alternative **paradigm** for better code structure and visibility. Also this concept primarily focus on:

1. Ability to model complex (real-world) objects
2. Code maintainability
3. Code re-useability and modularization (readability)

In simple consideration:

- the concept of OOP is an **advanced** form of **encapsulation** similar to functions. Yet, the abstraction of OOP paves the way for **parallel computing** and hence enables handling complex and resource intense problems.
- all instructions (code) are still executed **sequentially**.
- OOP provides an option to **restrict** and **specify** manipulation by accessing elements through **interfaces**.

Object-Orientated Programming: What is it?

Object-Orientated Programming means:

- pack particular functions into a box,
- and personalize the box with specific parameters.

The box has **always** the same **toolbox** and **set of properties**. Yet, the property values are box specific (i.e. Length, width).

The box can be reproduced using assembly instructions.

```
def mean_item(items):
    avg = 0
    for n,item in enumerate(items):
        avg += item
    else:
        return avg/n
```

```
def check_items(items, th):
    for item in items:
        if item > th:
            return False
        elif item < th:
            return True
        else:
            return "undefined"
```

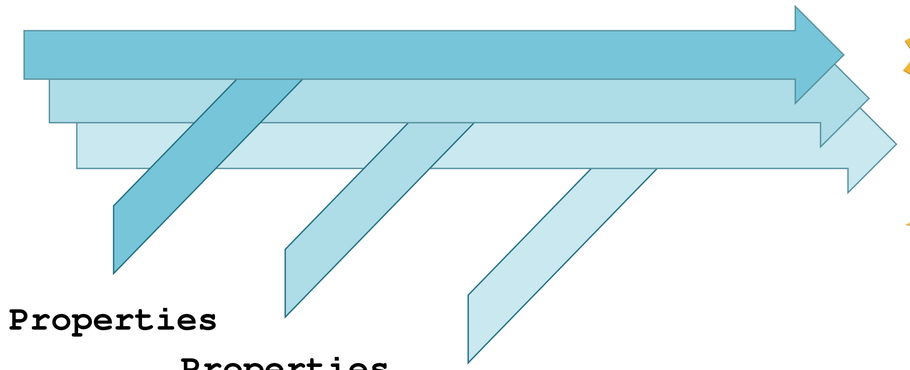
```
def open_box(box):
    with open(box) as f:
        return f.read()
```

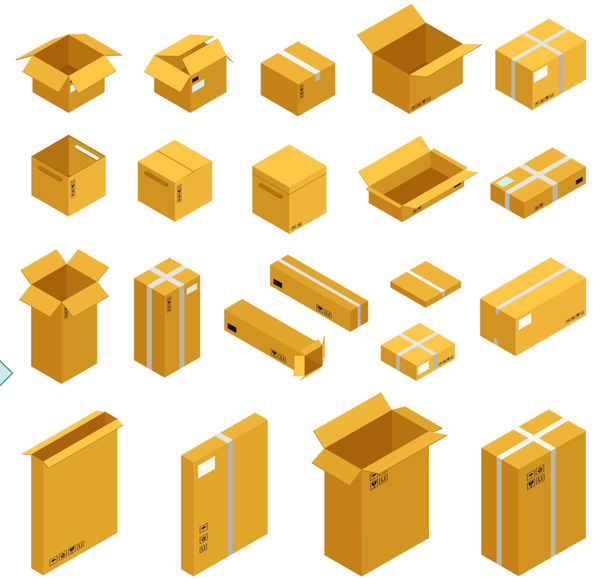
Object-Orientated Programming: Classes and Objects

Assembly instructions for box construction are represented by the **class**. And the produced boxes, all with the same set of tools but different property values, are called **objects**.

```
class Package:

      package_nr = 0

      def __init__(self, length, width,
                   height, weight):
            self. height = height
            self. width = width
            self. length = length
            self. weight = weight
            self. package_nr =\
                  Package.package_nr + 1
            package_nr += 1

      def __str__(self):
            return (f"Package number: "
                    f"{self.package_nr}")

      def get_weight(self):
            return self.weight
```

Class





Object(s)

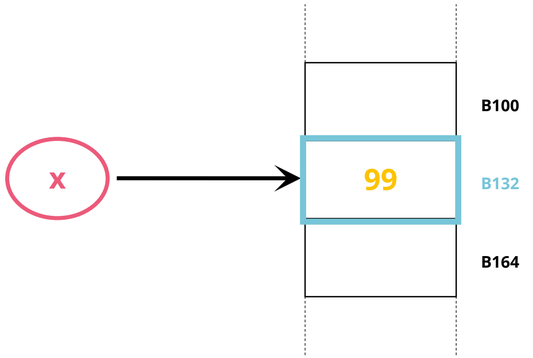
Object-Orientated Programming: Classes and Objects

In Python, everything is an object and variables hold references to these objects. This means, when you assign an object to a variable, you actually create a reference to that object in memory.

Consequently, **all data types** including string, int, double, list, etc. are **objects**.

- By assigning a value to a variable, Python implicitly creates a new object with this value.

```
x = 99                          # object of class int
s = 'hello world'               # object of class str
li = [1, 2, 3, 4]               # object of class list
```



And accessing objects requires a unique reference. This means, referring to the desired **namespace** and **object** using the **dot-notation**.

```
object.attribute                # accessing an attribute of an object
Class.attribute                 # accessing an attribute of a class
```

Object-Orientated Programming: Class Components

A class in python consist of several elements:

- **header** with class name
  > indicates the beginning of a class description.
- **__init__()** method: the constructor
  > constructs a new object of the class.
- **variables**
  > class: variables belonging to the class description.
  > object: variables belonging to a particular object.
- **methods (functions)**
  > class: methods provided by the class description.
  > object: methods provided by the object.
- **`self`** attribute
  > reference to the object: required to access object attributes.

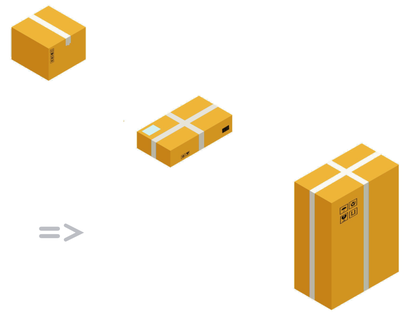
```
class Package:

    package_nr = 0

    def __init__(self, length, width,
                 height, weight):
        self.height = height
        self.width = width
        self.length = length
        self.weight = weight
        self.package_nr =\
            Package.package_nr + 1
        Package.package_nr += 1

    def __str__(self):
        return (f"Package number: "
                f"{self.package_nr}")

    def get_weight(self):
        return self.weight
```

Object-Orientated Programming: Class Components – constructor (__init__())

Generating a new object, requires calling the class name and defining the specific properties by attributes passed to the constructor:

```
my_package = Package(length=33, width=27.2, height=25, weight=13.7)
```

```
package_1 = Package(25, 25, 18, 9.7)          =>
package_2 = Package(33, 19, 5, 3.2)           =>
package_3 = Package(33, 19, 37, 13.7)         =>
```



Each time when generating a new object, python calls the constructor and passes the “properties” (attributes) to the **__init__()** method like: `__init__(length=33, width=27.2, height=25, weight=13.7)`

Object-Orientated Programming: Class Components – methods and variables

The class definition fully describes the class (i.e. the object). It comprises a **mix** of **all** variables and methods provided for the **whole class** and **particular objects**.

Objectives:

- Object methods (and variables) belong to a particular object.
- **Each** object of the same class provides **all the same** object methods.
- Object methods can access object specific variables (restricted visibility).

Solution:

- Object methods (and variables) are defined in the class definition **only once**.
- Methods and variables are called with the object or by reference (`self`-keyword), respectively.

Object-Orientated Programming: Class Components – `self` keyword

The `self` keyword points to the object it belongs to (namely itself).

- It allows to call or apply a variable and method, respectively, of (on) a particular object.

Consequently:

1. an object method (also `__init__()`) **always** requires a reference to the object by the keyword **`self`**.

```
def my_obj_fun(self, *param1, param2, param3)
```

2. object variables are always referred to by the keyword **`self`**.

```
self.my_obj_var = “hello world”
```

The `self` ‘keyword’ is **not** a regular keyword and can be **any word**. But it is a must have parameter!

Class Syntax

Class Syntax: Class Header and Documentation

Equally to control structures and functions, the class is introduced by means of a header that is terminated with a column (:).

The header starts with the keyword: **`class`**, followed by the **class name** and the column (**:**).

```
class Package:
    """
    This is the documentation of the
    class Package. It is not very short
    and spans multiple lines.
    """
```

- The header indent declares the block the class is visible in.
- Classes can be nested in a if-block or a subclass.
- The keyword must be lower case.
- In Python, class names follow the PascalCase convention (each word begins with a capital letter).
- When a class definition is entered, a new namespace is created.
- The class documentation directly follows after the header and is defined as a multi-line text block.
- Docu: https://docs.python.org/3/tutorial/classes.html#class-definition-syntax

Class Syntax: Class Variables

Class variables belong to a class. This are ‘global’ variables storing information superior of all instances of a class.

A class variable…

- … follows the naming convention of regular variables.
  `class_variable`
- … is declared right after the class header usually.
  `class_variable = ‘hello world’`
- … requires **no reference** for declaration **in class**.
  ~~**keyword**~~ `class_variable = ‘hello world’`
- … is always called with the class reference **.**
  `Class``.class_variable`
- … can be created on runtime (i.e. without declaration in class).
  `Class.class_variable = 66`

```
class Package:
    """
    This is the documentation of the
    class Package. It is not very short
    and spans multiple lines.
    """
    package_nr = 0

    def __init__(self, length, width,
                 height, weight):
        Package.package_nr += 1
```

Class Syntax: Constructor

The constructor (`__init__()` method) is called by Python **each time** when a new object is generated and defines the **properties** of the new object.

```
package_1 = Package(25, 25, 18, 9.7)
```

The `__init__` method (function) ...

- ... is defined with the keyword `def` and terminated with columns (:)
  `def` `__init__(self, param1, ...):`
- ... always has the name **`__init__`**`(...)`
- ... requires a reference to the object itself with the `self` parameter.
  `def __init__(`**`self`**`, param1, ...)`
- ... requires the `self` object reference always at the **first** position.
- ... primarily defines the object properties, i.e. the **object** variables.
  `self.obj_var = 'hello world'`

```
class Package:
    """
    This is the documentation of the
    class Package. It is not very short
    and spans multiple lines.
    """
    package_nr = 0

    def __init__(self, length, width,
                 height, weight):
        self. height = height
        self. width = width
```

Class Syntax: Object Variables

Object variables belong to the object and define the properties of it.

An object variable…

- … follows the naming convention of regular variables.
  `object_variable`
- … is declared in the **constructor** or any other **method**.
- … always requires the object reference, for declaration.
  `self.object_variable = ‘hello world’`
- … always requires the object reference, for call:
  `self.object_variable`
  `object.object_variable`
- … can be created on runtime (i.e. without declaration in class).
  `object.object_variable = 66`

```
class Package:
    """
    This is the documentation of the
    class Package. It is not very short
    and spans multiple lines.
    """
    package_nr = 0

    def __init__(self, length, width,
                 height, weight):
        self. height = height
        self. width = width
        self. length = length
        self. weight = weight
```

Class Syntax: Methods

Equally to variables, methods can belong to an object or a class itself. The declaration and call is the same as for regular functions, yet always requires a reference.

Object method:

Declaration	`def obj_method(self, *param1, param2, param3=True)`
Call class intern	`self.obj_method(1, 2, 3, 4)`
Call by object	`object.obj_method(1, 2, 3, 4)`
Call by class	`Class.obj_method(object, 1, 2, 3, 4)`

Class method:

Declaration	`@classmethod` `def cls_method(cls, *param1, param2, param3=True)`
Call by class	`Class.cls_method(1, 2, 3, 4)`
Call by object	`object.cls_method(ClassName, 1, 2, 3, 4)`

Class Syntax: Methods

Calling an object or class method always requires a **reference**. As long as the method is called by addressing the object or class, Python implicitly passes the reference to the parameter list.

Object method:

Assuming the following object method is declared and called:

```
def obj_method(self, *param1, param2, param3=True)

self.obj_method(1, 2, 3, 4)
object.obj_method(1, 2, 3, 4)
```

Class method:

Assuming the following class method is declared and called:

```
@classmethod
def cls_method(cls, *param1, param2, param3=True)

Class.cls_method(1, 2, 3, 4)
```

Print an Object

Information Technology

April 2, 2025

Print an Object: The `__str__()` Method

Printing a string to the console is intuitive and equally to write text on screen. However, how is it with objects?

- Why is a **number** printed when **`int`** is passed to the `print` function?
- Why is **`True`** or **`False`** printed when a **boolean** is passed to the `print` function?
- Why is **`range(x, y)`** printed when a **range** object is passed to the `print` function?
- Why is s.th. the object **location** printed when an object like **`iter`** is passed to the `print` function?

Because print calls the object method `__str__()` and expects a `str` object that it prints to the console.

Consequently, by implementing a `__str__()` method, the programmer can define what is printed when an object of an own class is passed to the `print` function.

```
def __str__(self):
    return ('object name')
```

HSLU Lucerne University of Applied Sciences and Arts

School of Computer Science and Information Technology
Research
Ramón Christen
Research Associate Doctoral Student

Phone direct +41 41 757 68 96
ramon.christen@hslu.ch

HSLU T&A, Competence Center Thermal Energy Storage
Research
Andreas Melillo
Lecturer

Phone direct +41 41 349 35 91
andreas.melillo@hslu.ch