

Curso python (PCPP1) PUE 2023

python

- Jordi Ariño
 - jordi.arino@pue.es
 - jordiAS2K@gmail.com
- Plataforma: EDUBE
- Recursos
 - Referencia: <https://docs.python.org/3/reference/index.html>
 - Librerías: <https://docs.python.org/3/library/index.html>
 - funciones built-in: <https://docs.python.org/3/library/functions.html>
 - PEP8: <https://peps.python.org/pep-0008/>
 - Paquetes: <https://pypi.org/>

PCPP1 / M1 | Advanced Perspective of Classes and Object-Oriented Programming in Python

- Classes, instances, attributes, methods, as well as working with class and instance data;
- shallow and deep operations;
- abstract classes, method overriding, static and class methods, special methods;
- inheritance, polymorphism, subclasses, and encapsulation;
- advanced exception handling techniques;
- the pickle and shelve modules;
- metaclasses.

1 OOP FOUNDATIONS

- [1.1 Classes, Instances, Attributes, Methods — introduction](#)
- [1.2 Working with class and instance data – instance variables](#)

2 OOP ADVANCED

- [2.1 Python core syntax](#)
- [2.2 Inheritance and polymorphism — Inheritance as a pillar of OOP](#)
- [2.3 Extended function argument syntax](#)
- [2.4 Decorators](#)
- [2.5 Different faces of Python methods](#) ← resumen
- [2.6 Abstract classes](#)
- [2.7 Encapsulation](#)
- [2.8 Composition vs Inheritance - two ways to the same destination](#)
- [2.9 Inheriting properties from built-in classes](#)

3 ADVANCED TECHNIQUES OF CREATING AND SERVING EXCEPTIONS

- [3.1 Advanced techniques of creating and serving exceptions](#)

4 OBJECT PERSISTENCE

- [4.1 Shallow and deep copy operations](#)

- [4.2 Serialization of Python objects using the pickle module](#)
- [4.3 Making Python objects persistent using the shelve module](#)

5 METAPROGRAMMING

- [5.1 Metaprogramming](#)

PCPP1 / M2 | Best Practices and Standardization (Python Enhancement Proposals)

- PEP 20 (The Zen of Python)
- PEP 8 (Style Guide for Python Code)
- PEP 257 (Docstring conventions)
- how to avoid common errors and mistakes when writing code;
- how to write elegant and effective code.

1 INTRODUCTION TO PEPS

- [1.1 What is PEP?](#)
- [2.1 PEP 20 – The Zen of Python](#)
- [3.1 PEP 8 – Introduction](#)
- [4.1 PEP 257 – Docstring Conventions](#)

PCPP1 / M3 | Introduction to GUI Programming in Python: TkInter (Python Enhancement Proposals)

- what GUI is and where it came from;
- how to construct a GUI using basic blocks and conventions;
- how event-driven programming works;
- some popular and commonly used GUI environments and toolkits;
- what tkinter is and how to build a GUI with its help;
- how to use widgets, windows, and events, and how to create basic applications based on tkinter's application life cycle.

1 GUI PROGRAMMING

- [Python Professional Course Series: GUI Programming](#)
- [1.2 Let TkInter speak!](#)
- [1.3 Settling widgets in the window's interior](#)
- [1.4 Coloring your widgets](#)
- [1.5 A simple GUI application](#)
- [1.6 Events and how to handle them](#)
- [1.7 Visiting widgets' properties](#)
- [1.8 Interacting with widget methods](#)
- [1.9 Looking at variables](#)
- [2.1 A small lexicon of widgets - Part 1](#)
- [2.2 A small lexicon of widgets - Part 2](#)
- [2.3 A small lexicon of widgets - Part 3](#)
- [2.4 Shaping the main window and conversing with the user](#)

- [2.5 Working with the Canvas](#)

PCPP1 / M4 | Working with RESTful APIs

- the basic concepts of network programming, REST, network sockets, and client-server communication;
- how to use and create sockets in Python, and how to establish and close the connection with a server;
- what JSON and XML files are, and how they can be used in network communication;
- what HTTP methods are, and how to say anything in HTTP;
- how to build a sample testing environment;
- what CRUD is;
- how to build a simple REST client, and how to fetch and remove data from server, add new data to it, and update the already-existing data.

1

- [1.1 Python Professional Course Series: RESTful APIs](#)
- [1.2 How to use sockets in Python](#)
- [1.3 JSON - our new friend](#)
- [1.4 Talking to JSON in Python](#)
- [1.5 What is XML and why do we prefer to use JSON?](#)
- [1.6 Making life easier with the requests module](#)
- [1.7 Four magic letters: CRUD](#)

summary

- class, instance, object, attribute, method, type (`__class__`)
- instance variables: `<object>.__dict__`
- class variables:
 - `<class>.__dict__`
 - `<class>.class_var = 1`
- superclass methods:
 - `__init__`
- special methods:
 - `__name__`
 - `__getattr__`
- magic methods
 - <https://docs.python.org/3/reference/datamodel.html#special-method-names>
 - `dir()`
 - `help()`
- Inheritance
 - MRO $\equiv$ Method Resolution Order
- Polymorphism
- arguments
 - `*args`: tuple (a,b,c)
 - `**kwargs`: dictionary {}
- decorators
 - `@simple_decorator()`

```
class Classe:
    class_var = 0
    def __init__(self):
        self.instance_var = 0
```

```
Classe.class_var += 1
```

```
class SubClasse(Classe):  
    pass
```

```
def simple_decorator(own_func):  
    def internal_wrapper(*args,**kwargs):  
        print("pre-execution")  
        own_function(*args,**kwargs) # original method executed  
  
    return internal_wrapper
```

```
@simple_decorator  
def decorators(*args,**kargs):  
    pass
```

```
def simple_decorator_attributes(attribute):  
    def wrapper(our_func):  
        def internal_wrapper(*args):  
            pass  
        return internal_wrapper  
    return wrapper
```

```
@simple_decorator_attributes('argument')  
def decorators_with_attribute(*args):  
    pass
```

```
@simple_decorator  
@simple_decorator_attributes('other_argument')  
# simple_decorator( simple_decorator_attributes( funcion() ) )
```

```
class SimpleDecoratorClass:  
    def __init__(self, own_func):  
        self.func = own_func  
    def __call__(self,*args,**kwargs):  
        print(self.func.__name__)  
        self.func(*args,**kwargs)
```

```
class SimpleDecoratorClassWithArguments:  
    def __init__(self,argument):  
        pass  
    def __call__(self,own_func)
```

```
@SimpleDecoratorClass  
def simple_function(*args,**kwargs):  
    pass
```

```
def object_counter(class_):  
    class_.__getattr__orig = class_.__getattr__  
  
    def new_getattr(self, name):  
        if name == 'mileage':  
            print('We noticed that the mileage attribute was read')  
        return class_.__getattr__orig(self, name)
```

```
class_.__getattr__ = new_getattr
return class_

@object_counter
class Car:
    def __init__(self, VIN):
        self.mileage = 0
        self.VIN = VIN
```

otros conocimientos

- [yield](#)
 - [generadores](#)
- logs: <https://atareao.es/pyldora/tus-logs-en-python-de-forma-eficiente/>

guia estilo

- [PEP20](#)
- `pip install pycodestyle`

utilidades

- <https://sqlitestudio.pl/>
- multiplataforma? BeeWare: <https://docs.beeware.org/en/latest/index.html>

From:

<https://miguelangel.torresegea.es/wiki/> - miguel angel torres egea

Permanent link:

<https://miguelangel.torresegea.es/wiki/info: cursos:pue:python-pcpp1?rev=1705660645>

Last update: **19/01/2024 02:37**

