

Capacitación Semillero DISW

Python Clase 1

Docentes: Lautaro Delgado, Idc0295@gmail.com
Magdalena Bouza, bouza.magdalena@gmail.com

Cronograma



Temario

- **Clase 1:**
 - Instalación de Python y herramientas (Anaconda).
 - Preparación del entorno
 - IDE + editor de código
 - Tipos de datos, bucles, funciones
 - OOP en Python
- **Clase 2:**

Librerías: Numpy, Pandas, Matplotlib, Seaborn
- **Clase 3:**

Puesta en producción del código: diseño de una librería/API, haciendo uso de repositorio Git. Unit Testing con PyTest



Python

Clase 1

Anaconda



Anaconda: Instalación

- ▷ Linux
- ▷ Windows
- ▷ Mac

Anaconda: Overview

- ▷ Conda
- ▷ Anaconda Navigator
- ▷ Scientific Packages

Conda

- Creación de Librerías
- Creación de Environments

IDEs y editores de código

Entorno de desarrollo integrado (IDE)

Un *entorno de desarrollo integrado* (IDE) es una aplicación que integra distintas herramientas necesarias para el desarrollo de software.

En general consta de:

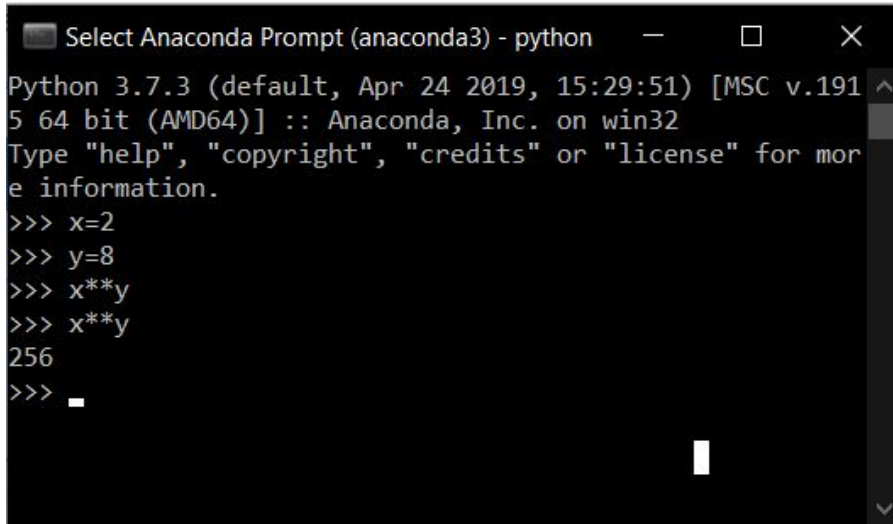
- ▷ editor de código
- ▷ herramientas de construcción automáticas
- ▷ depurador
- ▷ (opc) compilador y/o intérprete

[Instalación de Pycharm](#)

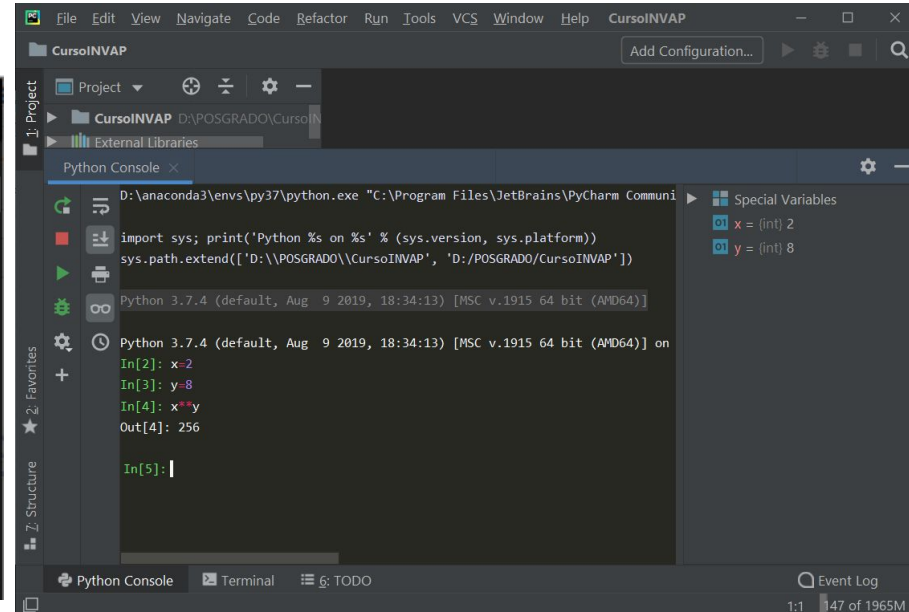


Intérprete de Python (shell)

Aquí podemos correr directamente sentencias de Python.



```
Select Anaconda Prompt (anaconda3) - python
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.191
5 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for mor
e information.
>>> x=2
>>> y=8
>>> x**y
>>> x**y
256
>>> .
```



```
File Edit View Navigate Code Refactor Run Tools VCS Window Help CursoINVAP
CursoINVAP
Add Configuration...
Project
CursoINVAP D:\POSGRADO\CursoINVAP
External Libraries
Python Console
D:\anaconda3\envs\py37\python.exe "C:\Program Files\JetBrains\PyCharm Communi
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['D:\POSGRADO\CursoINVAP', 'D:\POSGRADO\CursoINVAP'])
Python 3.7.4 (default, Aug 9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)]
Python 3.7.4 (default, Aug 9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)] on
In[2]: x=2
In[3]: y=8
In[4]: x**y
Out[4]: 256
In[5]:
Special Variables
x = (int) 2
y = (int) 8
Python Console Terminal TODO Event Log
1:1 147 of 1965M
```

Código (script)

Son archivos con extensión .py donde vamos a escribir las distintas sentencias que queremos correr en nuestro código.

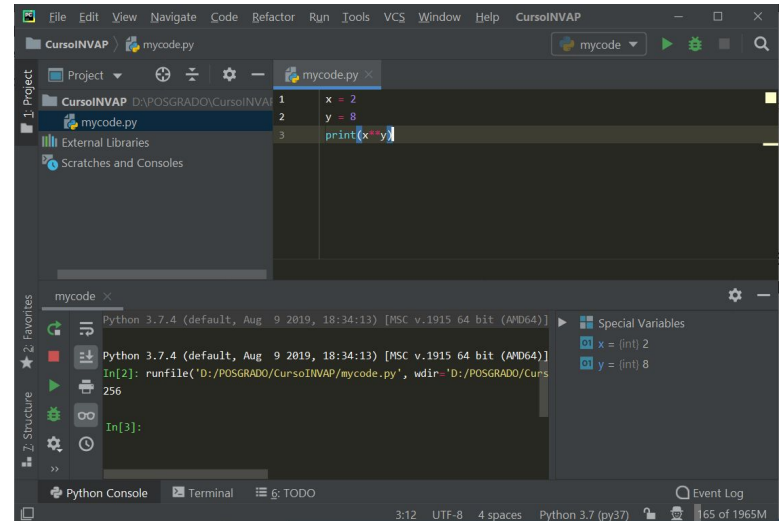
```
Anaconda Prompt (anaconda3)

(base) C:\Users\magui>d:

(base) D:\>cd POSGRADO\CursoINVAP\

(base) D:\POSGRADO\CursoINVAP>python mycode.py
256

(base) D:\POSGRADO\CursoINVAP>
```



Tipos y estructuras de datos



Tipos de datos

- ▷ Números:
 - Enteros (integer): 2, 5, 155
 - Punto flotante (float): 1.5, 5.36687
 - complejos (complex): 1-j5
- ▷ Booleanos (bool): True/False
- ▷ Texto (string): 'hola'

Estructuras de datos

- ▷ Listas: Listas de valores. Cada valor está indexado comenzando por el índice 0. Se pueden eliminar, agregar y modificar los valores. `[1, 4, 5, 6, 7]`
- ▷ Tuplas: similares a las listas, pero no se pueden modificar. `('ene', 'feb', 'mar', 'abr', ..., 'dic')`
- ▷ Diccionarios: Se tiene un conjunto de índices (keys), y cada índice tiene asociado un valor. Los valores del diccionario no están numerados. Se puede agregar, quitar y modificar entradas del diccionario.
`{ 'Pedro':1.73, 'Ana':1.58, 'Sofia':1.67 }`

Estructuras de datos

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

Estructuras de datos

Operation	Average Case
Copy	$O(n)$
Append[1]	$O(1)$
Pop last	$O(1)$
Pop intermediate	$O(k)$
Insert	$O(n)$
Get Item	$O(1)$
Set Item	$O(1)$
Delete Item	$O(n)$
Iteration	$O(n)$
Get Slice	$O(k)$
Del Slice	$O(n)$
Set Slice	$O(k+n)$
Extend[1]	$O(k)$
 Sort	$O(n \log n)$
Multiply	$O(nk)$
x in s	$O(n)$
min(s), max(s)	$O(n)$
Get Length	$O(1)$

Listas

Operation	Average Case	Amortized Worst Case
k in d	$O(1)$	$O(n)$
Copy[2]	$O(n)$	$O(n)$
Get Item	$O(1)$	$O(n)$
Set Item[1]	$O(1)$	$O(n)$
Delete Item	$O(1)$	$O(n)$
Iteration[2]	$O(n)$	$O(n)$

Diccionarios

Control de flujo



Operadores de comparación

Operator	Description	Example
<code>==</code>	Tests if two values are equal	<code>3 == 3</code>
<code>!=</code>	Tests that two values are <i>not</i> equal to each other	<code>2 != 3</code>
<code><</code>	Tests to see if the left-hand value is less than the right-hand value	<code>2 < 3</code>
<code>></code>	Tests if the left-hand value is greater than the right-hand value	<code>3 > 2</code>
<code><=</code>	Tests if the left-hand value is less than <i>or</i> equal to the right-hand value	<code>3 <= 4</code>
<code>>=</code>	Tests if the left-hand value is greater than or equal to the right-hand value	<code>5 >= 4</code>

Operadores lógicos

Operator	Description	Example
and &	Returns True if both left and right are true	(3 < 4) and (5 > 4)
or	Returns true if either the left or the right is true	(3 < 4) or (3 > 5)
not !	Returns true if the value being tested is False	not 3 < 2

if-else

Es una forma de programación condicional:

```
if condición:
```

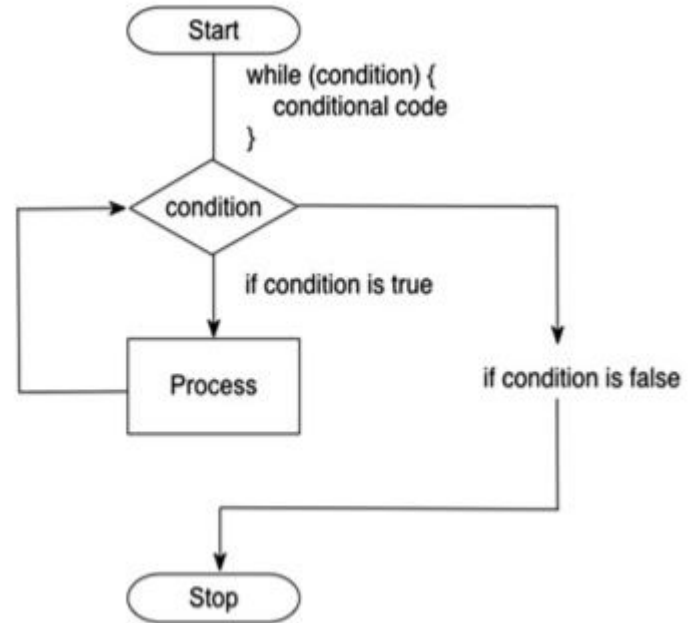
```
    sentencia
```

```
else:
```

```
    sentencia
```

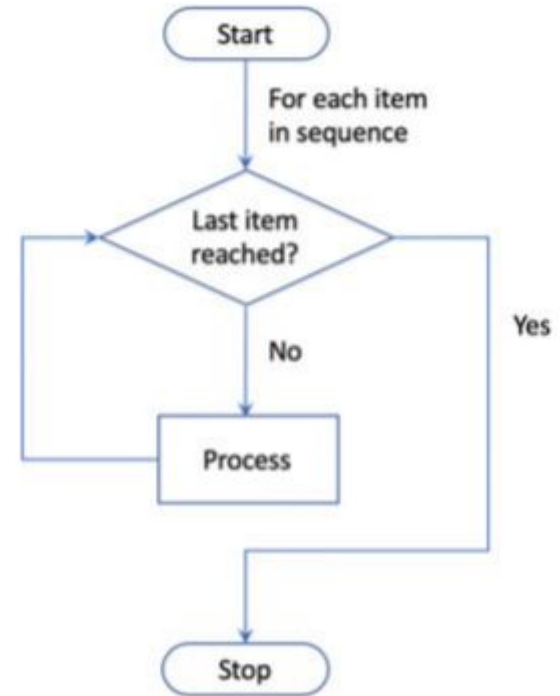
Ciclos while

```
while condición:  
    sentencia
```



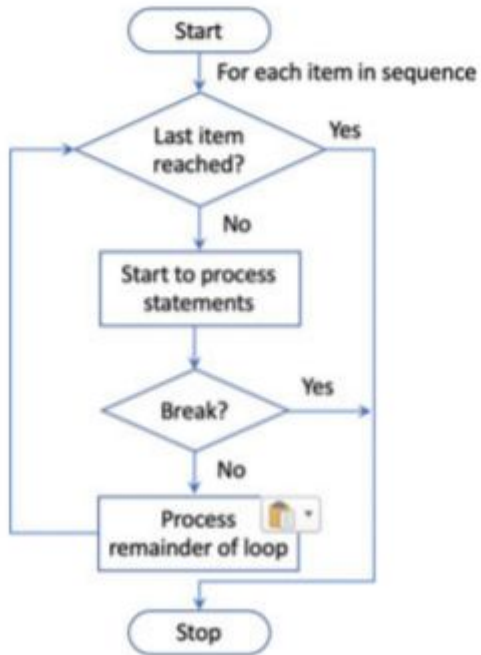
Ciclos for

```
for i in range(0,10):  
    print(i)
```

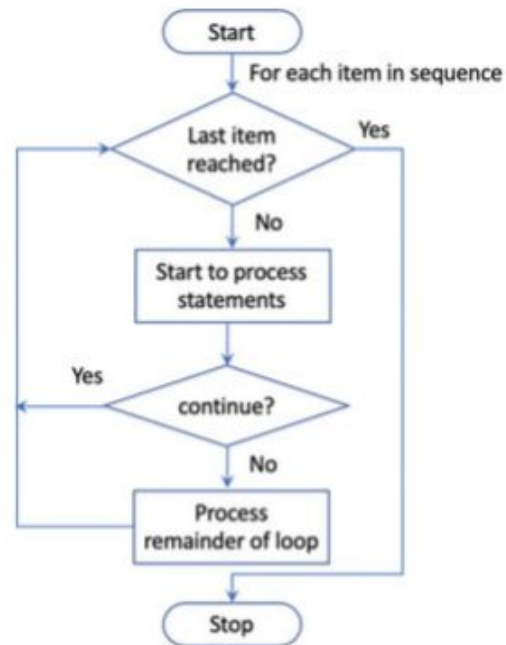


Alteración de Flujo

break



continue



Funciones

Funciones

```
def function_name(parameters) :  
    statement  
    ...  
    statement  
    (return _)
```

- ▷ built-in
- ▷ user defined

Funciones

- ▷ Múltiples parámetros
- ▷ Parámetros por defecto / opcionales
- ▷ Parámetros con nombre
- ▷ Argumentos posicionales (*args)
- ▷ Argumentos por palabras claves (**kwargs)

Funciones - Tipado

```
def function_name(a: str, b:List[Any],  
c:int=5) -> int:  
    """  
    (propósito de la función)  
    : param a: definición parámetro a  
    : return: definición salida  
    """  
  
    ...  
    return len(a)*c
```

Funciones lambda


Pueden tener cualquier número de argumentos pero únicamente una (1) expresión, que devuelve un valor:

```
func0 = lambda: print('no args')
```

```
func1 = lambda x: x * x
```

```
func2 = lambda x, y: x * y
```

Programación Orientada a Objetos (OOP)



¿Qué es OOP?

Es un paradigma de programación que nos permite organizar el código de una manera que se asemeja bastante a como pensamos en la vida real, utilizando clases. Estas nos permiten agrupar un conjunto de variables y funciones como veremos a continuación.

Clases

Es uno de los pilares de Python, y resulta un concepto central en la OOP.

Las **clases** funcionan como *templates* para construir instancias de una clase de objetos.

Un **objeto** es una instancia (o ejemplo) de una clase.

Las clases permiten al programador especificar la estructura de un objeto y su comportamiento.

Clases vs. Objetos

Clase

- ▷ Template para crear objetos
- ▷ Definir métodos (comportamientos comunes) para una clase de objetos
- ▷ Definir atributos (campos) dentro de los objetos
- ▷ Le puedo enviar mensajes

Objeto

- ▷ Son instancias de una clase
- ▷ Guardan los valores propios para una dada instancia
- ▷ Ejecutar métodos del objeto
- ▷ Puedo tener muchas copias (cada una con sus valores)
- ▷ Le puedo enviar mensajes

Terminología

- ▶ **Clase:** Define una combinación de datos y comportamientos que operan sobre esos datos
- ▶ **Instancia u objeto:** Es un ejemplo de una clase. Cada instancia de una clase tiene los mismos campos y atributos pero guarda sus propios valores
- ▶ **Atributos, campos, variables de instancia:** Los datos contenidos por un objeto se representan a través de sus atributos.
- ▶ **Método:** un procedimiento definido dentro de un objeto
- ▶ **Mensaje:** Se envía un mensaje al objeto requiriendo que ejecute cierta operación o acceder a cierto atributo

Cómo crear una clase

```
class MyClass():  
    def __init__(self,p1,p2..):  
        self.p1 = p1  
        self.p2 = p2  
        ...  
    def MyMethod1(self,):  
        statement  
        ...  
        statement  
    def MyMethod2(self,x1,):  
        statement  
        ...  
        statement  
        return variable
```

Método que inicializa los atributos de la clase.
p1 y p2 serán los atributos

Método 1: no tiene return.
Opera sobre los atributos de la clase

Método 2: Va a operar sobre el objeto, y finalmente devuelve un valor

Herencia

La herencia es el proceso por el cual una clase toma los atributos y métodos de otra.

Las clases más “nuevas” se llaman clases hijo (*child class*), mientras que la clases de las que la clase hijo hereda las propiedades se denomina clase padre (*parent class*).

La clase hijo puede sobrescribir o extender los atributos y métodos de la clase padre

Ejemplo de herencia

```
class MyParent():
    def __init__(self,p1,p2..):
        self.p1 = p1
        self.p2 = p2
    ...
    def MyMethod(self,):
        statement
    ...
    statement
    def MyMethod2(self):
        pass
class MyChild(MyParent):
    def MyMethod2(self,x):
        statement
    ...
    return var
```

Tipo de Métodos

- ▷ Métodos de instancia
- ▷ Métodos de clase
- ▷ Métodos estáticos

```
class Clase:
    def metodo(self):
        return 'Método normal', self

    @classmethod
    def metododeclase(cls):
        return 'Método de clase', cls

    @staticmethod
    def metodoestatico():
        return "Método estático"
```

Properties

- ▷ Encapsulamiento de atributos
- ▷ Métodos getter, setter

Python Data Model

Modificación básica de clases:

→ “dunder methods”: `__init__`, `__add__`, `__repr__`, etc.

Referencia completa: [Data Model](#)