Python

Ing. Pedro Parra
Diplomado de Programación

Versión de Python que usaremos:
3.11.2 en adelante

Instaladores:
https://www.python.org/downloads/

Documentación sobre Python

1) Libro: Python para todos. **Autor:** Raúl González Duque

2) Web: https://wiki.python.org/moin/SpanishLanguage

3) Github: https://github.com/Asabeneh/30-Days-Of-Python

4) Psycopg: https://www.psycopg.org/docs/

¿Qué es Python?

"Se trata de un lenguaje interpretado o de script, con tipado dinámico, fuertemente tipado, multiplataforma y orientado a objetos".

(González Duque Raúl, Python para todos, 2023, p. 7).

Lenguaje interpretado

"Un lenguaje interpretado o de script es aquel que se ejecuta utilizando un programa intermedio llamado intérprete, en lugar de compilar el código a lenguaje máquina que pueda comprender y ejecutar directamente una computadora (lenguajes compilado)".

"… tiene, no obstante, muchas de las características de los lenguajes compilados, por lo que se podría decir que es semi interpretado"

(González Duque Raúl, Python para todos, 2023, p. 7).

Tipado Dinámico

"La característica de tipado dinámico se refiere a que no es necesario declarar el tipo de dato que va a contener una determinada variable sino que su tipo se determinará en tiempo de ejecución según el tipo del valor al que se asigne, y el tipo de esta variable puede cambiar si se le asigna un valor de otro tipo."

(González Duque Raúl, Python para todos, 2023, p. 8).

Fuertemente tipado

"No se permite tratar a una variable como si fuera de un tipo distinto al que tiene, es necesario convertir de forma explícita dicha variable al nuevo tipo previamente."

(González Duque Raúl, Python para todos, 2023, p. 8).

Ejemplo:

```
a=5
b="3"
c=a+b     #Error, b fue declarado como cadena
```

Orientado a Objetos

"La orientación a objetos es un paradigma de programación en el que los conceptos del mundo real relevantes para nuestro problema se trasladan a clases y objetos en nuestro programa. La ejecución del programa consiste en una serie de interacciones entre los objetos."

(González Duque Raúl, Python para todos, 2023, p. 8).

Configurando nuestro entorno de Desarrollo

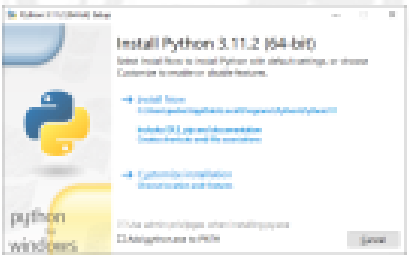
Debemos tener instalado el siguiente editor de código:

- ✓ Visual Studio Code

Y los siguientes plugins instalados en Visual Studio Code: Python(Microsoft),Prettier, Gitlens, Intellicode

Instalación de Python

Descargamos el instalador para nuestra computadora desde https://www.python.org/downloads/



El proceso de instalación en sencillo, sólo debemos seguir las instrucciones del instalador.

Nuestro Primer Programa

Objetivo: aprender la sintaxis básica de programación en Python utilizando el esquema de Ubicación para conocer los siguientes conceptos:

- Crear comentarios de una o múltiples líneas
- Declarar variables y tipos de datos
- Funciones básicas
- Inferencia de datos
- Operadores lógicos
- Condicionales

Ejercicio de Refuerzo: crear y administrar un repositorio de aprendizaje de Python usando git

Creando y Configurando el Repositorio

• Desde visual studio code entramos en la consola (Ctrl + Shift + ñ) y seleccionamos git bash.

• Creamos una carpeta donde programaremos el código fuente de todos los ejercicios.

```
mkdir aprendiendo_pyhon
```

• Entramos en la carpeta

```
cd aprendiendo_python
```

Creando y Configurando el Repositorio

• Desde visual studio code entramos en la consola (Ctrl + Shift + ñ) y seleccionamos git bash.

• Creamos una carpeta donde programaremos el código fuente de todos los ejercicios.

```
mkdir aprendiendo_pyhon
```

• Entramos en la carpeta

```
cd aprendiendo_python
```

Creando y Configurando el Repositorio

- git config –global user.email pedro.parra82@gmail.com
- git init

Repasando antes de Programar

CARACTER	CODIGO ASCII
Llave Apertura {	**Alt+123**
Llave Cierre }	**Alt+125**
Corchete apertura [	**Alt+91**
Corchete Cierre]	**Alt+93**
Paréntesis Apertura (	**Alt+40**
Paréntesis Cierre)	**Alt+41**
Signo Mayor que >	**Alt+62**
Signo Menor que <	**Alt+60**

Repasando antes de Programar

CARACTER	CODIGO ASCII
Arroba @	**Alt+64**
Dos Puntos :	**Alt+58**
Punto y Coma ;	**Alt+59**
Letra Ñ	**Alt+165**
Letra ñ	**Alt+164**
Numeral #	**Alt+35**
Guión -	**Alt+45**
Guión Bajo o Piso _	**Alt+95**
Sombrerito ^	**Alt+94**

Repasando antes de Programar

CARACTER	CODIGO ASCII
Asterisco *	**Alt+42**
Barra Invertida \	**Alt+92**
Barra Vertical |	**Alt+124**
Barra /	**Alt+47**
Ampersand o Y Comercial &	**Alt+38**
Comillas Dobles “”	**Alt+34**
Comillas Simples ''	**Alt+39**
Negación ¡	**Alt+33**

Primer Ejercicio

El objetivo general de este ejercicio es aprender a usar la sintaxis básica del lenguaje Python.

Objetivo 1: Crear comentarios de una o múltiples líneas
Objetivo 2: Declarar variables y tipos de datos
Objetivo 3: Funciones básicas
Objetivo 4: Inferencia de datos
Objetivo 5: Operadores lógicos
Objetivo 6: Condicionales

Empezamos a Programar

- La extensión de un archivo en Python es .py
  Ejemplo: ejercicio1.py

- Para crear comentarios de una sola línea usamos: #

```
Ejemplo:
#Este será nuestro primer ejercicio de Python en
Ingeniería Digital
```

- Para crear comentarios de varias líneas usamos: “””

```
"""Los objetivos de este ejercicio son:
Objetivo 1:  Crear comentarios de una o múltiples
líneas
Objetivo 2:  Declarar variables y tipos de datos
Objetivo 3:  Funciones básicas
Objetivo 4: Inferencia de datos
Objetivo 5:  Operadores lógicos
Objetivo 6: Condicionales
"""
```

Tipos de Datos Básicos

```
"""Objetivo 2: los tipos de datos mas básicos son:
-    Enteros, Reales(números con decimales), Cadenas(Textos) y
Booleanos. """

#Declarando variables enteras
#Número de continentes es un número entero pero no tenemos que
especificarlo
nro_con=6    #no usamos ; al final
nro_pai=195 #Número de países del mundo
nro_est=23   #Número de estados de Venezuela
nro_ciu=100 #Número de ciudades en Venezuela con mas de 50.000
Habitantes
nro_zon=1    #Venezuela tiene 1 zona horaria


#Declarando variables reales.
pro_vid=71.1  #Promedio de esperanza de vida en Venezuela
tas_cam=25.34 #Tasa de Cambio con respecto al dólar
por_imp=16.00
```

Declarando Cadenas

```
#Declarando cadenas(Textos).
nom_pai="Venezuela" #Las cadenas deben llevar comillas
cap_pai='Caracas'  #Pueden ser comillas simples o dobles


#Puedo usar comillas triples para asignar textos de
múltiples líneas

des_pai="""Venezuela es un país de la costa norte de
América del Sur, con diversas atracciones naturales. A lo
largo de su costa en el Caribe,hay islas turísticas
tropicales, entre ellas la Isla de Margarita y el
archipiélago Los Roques. Al noroeste está la cordillera de
los Andes"""
```

Declarando Cadenas

```
#Declarando Booleanos (True o False)
#Clasificado al mundial de beisbol: Verdadero(True)
mun_bei=True

#Clasificado al mundial de beisbol: Falso(False)
mun_fut=False

"""También puedo declarar varias variables en una sola
línea, aunque no es muy recomendado. Ejemplo:"""

num_hab, cti_pai, fec_ind = 30000000, "+58", 1811
```

Funciones Básicas

```
"""Objetivo 3: Funciones Básicas:
- print: imprimir mensajes en pantalla
- input: pedir datos al usuario
- float: convertir un string a real
"""

print("Bienvenidos a Python") #Mensaje Sencillo
```

Inferencia de Datos

```
"""Objetivo 4: Inferencia de datos
Si necesito imprimir variables tenemos varias alternativas.
"""
#Alternativa 1
#Utilizamos el %s para las cadenas, %d para los estados
print("En suramérica se encuentra %s, cuya capital es %s y tiene %d
estados"%(nom_pai,cap_pai,nro_est))

#Alternativa 2
print("El promedio de vida en {} es de {} años.".format(nom_pai,pro_vid))

#Alternativa 3 - Recomendada
print(f"El número de habitantes de {nom_pai} es de {num_hab}")
print(f"La tasa de cambio para Venezuela es {tas_cam}")
```

Funciones Básicas

```
# Función input: pedir datos al usuario
print("Por favor ingrese a tasa de cambio para el dia de hoy")
tas_cam=input()
print(f"La tasa de cambio actualizada para Venezuela es {tas_cam}")


print("Por favor ingrese el precio mensual en Dólares del Diplomado de
Programación")
pre_dip=input()


print("Desea factura legal? S/N")
fac_leg=input().upper() #upper() convierte el texto en mayúsculas
```

Condicionales

```
#Objetivo 6: Condicionales

if fac_leg=="S":
    iva_vef=pre_vef*(por_imp/100) #Impuesto
    bas_imp=pre_vef-iva_vef  #Base Imponible
    print(f"""
    El precio del Diplomado en Bs es: {pre_vef}
    La base imponible es: {bas_imp}
    El impuesto a pagar es:  {iva_vef} """)
else:
    print(f"El  precio del Diplomado es de Bs: {pre_vef}")
```

Fin de Primer Ejercicio
Ejecutamos Commit

- `git add ejercicio1.py`
- `git commit -m “Ejercicio 1 – Python”`
- `git status`
- `git tag Ejercicio1`
- `git reflog`

Segundo Ejercicio

El objetivo general de este ejercicio es aprender a usar colecciones de tipos de datos.

Objetivo 1: Aprender a usar listas
Objetivo 2: Aprender a usar tuplas
Objetivo 3: Aprender a usar sets
Objetivo 4: Aprender a usar diccionarios

Objetivo 1: Aprender a usar listas ***Listas***

Es una colección ***ordenada*** y ***modificable*** de datos. Permite miembros ***duplicados***.

Sería equivalente a lo que en otros lenguajes se conoce por arrays, o vectores. Las listas pueden contener cualquier tipo de dato: números, cadenas, booleanos, … y también listas.

Listas

Crear una lista es tan sencillo como indicar entre corchetes, y separados por comas, los valores que queremos incluir en la lista.

Ejemplo:

paises = [“Venezuela”, “Colombia”, “Brasil” , “España” , “Argentina”, “Italia”]

Podemos acceder a cada uno de los elementos de la lista escribiendo el nombre de la lista e indicando el índice del elemento entre corchetes.

Top 10
Funciones importantes sobre listas

```
paises = [“Venezuela”, “Colombia”,“Brasil”,“España”,“Argentina”,“Italia”]
```

- **print():** imprimir lista.
  `Ejemplo: print(“Paises: ”, paises)`

- **len():** contar ítems de una lista.
  `Ejemplo: print(“Número de Paises: ”, len(paises))`

- **append():** agregar un item a una lista al final de la lista.
  `Ejemplo: paises.append(“Uruguay”)`

- **insert():** insertar un item a una lista en un puesto específico
  `Ejemplo: paises. insert(3, “Australia”)`

- **remove():** remover la primera aparición de un item
  `Ejemplo: paises.remove(“Brasil”)`

Top 10
Funciones importantes sobre listas

• **pop():** remueve un índice específico, sino se especifica elimina el último registro.

```
Ejemplo: paises.pop(1) #elimina a Colombia
```

• **copy():** copia una lista

```
Ejemplo: copia=paises.copy()
```

• **count():** contar la cantidad de veces que aparece un elemento.

```
Ejemplo: paises.count("Venezuela")
```

• **index():** saber el índice del elemento buscado

```
Ejemplo: paises.index("Australia")
```

• **sort():** ordena la lista, si queremos ordenarla en reversa usamos reverse()

```
Ejemplo: paises.sort()
```

Objetivo 2: Aprender a usar listas ***Tuplas***

Una tupla es una colección de diferentes tipos de datos ordenados e inmutables (inmutables). Las tuplas se escriben con paréntesis, (). Una vez que se crea una tupla, no podemos cambiar sus valores. No podemos usar métodos de agregar, insertar en una tupla porque no es modificable (mutable). A diferencia de list, tuple tiene pocos métodos. Métodos relacionados con las tuplas.

Funciones importantes sobre Tuplas

```
continentes=("América","Asia","Europa","Antártida","Oceanía
","África")
```

- tuple(): crea una tupla vacía
  - Ejemplo: `continentes=tuple()`

- count(): cuenta la cantidad de veces que aparece un ítem en una tupla.
  Ejemplo: `continentes.count("Europa") # 1 Vez`

- index(): encuentra el índice de un elemento buscado
  Ejemplo: `continentes.index("África") #Indice 5`

- *len():* cuenta los ítems que posee la tupla.

Objetivo 3: Aprender a usar listas ***Sets***

Un set es una colección desordenada de elementos únicos, es decir, que no se repiten. Un set si puede ser modificable con las propiedades add(), update(), remove()

Funciones importantes sobre Sets

```
estados={"Táchira", "Mérida", "Trujillo", "Zulia",
"Lara", "Barinas"}
```

- set(): crea una tupla vacía
  Ejemplo: `estados=set()`

- add(valor): Agregando un nuevo valor al set
  Ejemplo: `estados.add("Falcón")`

- remove(valor): Eliminando un valor al set
  Ejemplo: `estados.remove("Zulia")`

```
#imprimiendo el set
   print("Los estados de Venezuela: ",estados)
```

Objetivo 4: Aprender a usar listas ***Diccionarios***

Un diccionario es una colección de tipos de datos no ordenados, modificables (mutables) emparejados (clave: valor).

Sintaxis:

```
x = {‘clave1’:’valor1', ‘clave2’:’valor2', ‘clave3’:’valor3', ‘clave4':’valor4'}
```

```
#Creando un diccionario complejo:
mundo = {
    'nro_con':6,
    'nro_pai':195,
    'nro_per':8000000000,
    'continentes': ["América", "Asia",
"Europa","Antártida", "Oceanía","África"],
    'paises':{
        'Argentina':'Buenos Aires',
        'Colombia':'Bogotá',
        'Ecuador': 'Quito',
        'España': 'Madrid',
        'Venezuela':'Caracas'
            }
    }
```

Funciones importantes sobre Diccionarios

- dict(): crea un diccionario vacío
  Ejemplo: `mundo=dict()`

- keys(): Obtenemos las llaves del diccionario.
  Ejemplo: `mundo.keys()`

- values(): Obtenemos las llaves del diccionario.
  Ejemplo: `mundo.values()`

- get(): Acceder a un elemento por nombre de clave genera un error si la clave no existe. Para evitar este error podemos usar el método get
  Ejemplo: `mundo.get('pais')`

Fin de Segundo Ejercicio
Ejecutamos Commit

- `git add ejercicio2.py`
- `git commit -m “Ejercicio 2 – Python”`
- `git status`
- `git tag Ejercicio2`
- `git reflog`

Tercer Ejercicio

El objetivo general de este ejercicio es aprender a usar los ciclos

Objetivo 1: Aprender a usar el ciclo while

Objetivo 2: Aprender a usar el ciclo for

Objetivo 1: Aprender a usar el Ciclo While

Mientras que los condicionales nos permiten ejecutar distintos fragmentos de código dependiendo de ciertas condiciones, los bucles nos permiten ejecutar un mismo fragmento de código un cierto número de veces, mientras se cumpla una determinada condición

Ejemplo

```
paises= ["Venezuela", "Colombia", "Brasil" , "España" ,
"Argentina",  "Italia"]

#contamos cuantos paises hay en la lista
nro_pai=len(paises)
con=0 #contador

while con < nro_pai:
    print("Imprimiendo el país Nro", con,paises[con])
    con=con+1
else:
#else en un while es opcional y algo interesante de Python
    print("Ya no hay mas países por imprimir")
```

Objetivo 2: Aprender a usar el Ciclo For…In

Una palabra clavefor se usa para hacer un bucle for, similar a otros lenguajes de programación, pero con algunas diferencias de sintaxis. Loop se usa para iterar sobre una secuencia (es decir, una lista, una tupla, un diccionario, un conjunto o una cadena).

Ejemplo 1

```
paises= ["Venezuela", "Colombia", "Brasil" ,
"España" , "Argentina",  "Italia"]

#Ejemplo 1
for nom_pai in paises:
     print("Paises de mi lista:", nom_pai)
```

Ejemplo 2

```
#Creando un diccionario complejo:
mundo = {
    'nro_con':6,
    'nro_pai':195,
    'nro_per':8000000000,
    'continentes': ["América", "Asia", "Europa","Antártida",
"Oceanía","África"],
    'paises':{
        'Argentina':'Buenos Aires',
        'Colombia':'Bogotá',
        'Ecuador': 'Quito',
        'España': 'Madrid',
        'Venezuela':'Caracas'
            }
    }
```

```
for dat_mun in mundo.values()
      print("Datos de mi diccionario",dat_mun)
```

Fin de Tercer Ejercicio
Ejecutamos Commit

- `git add ejercicio3.py`
- `git commit -m “Ejercicio 3 – Python”`
- `git status`
- `git tag Ejercicio3`
- `git reflog`

Cuarto Ejercicio

El objetivo general de este ejercicio es aprender a usar las funciones

Objetivo 1: Aprender a usar funciones sin parámetros

Objetivo 2: Aprender a usar funciones con parámetros fijos

Objetivo 3: Aprender a usar funciones con parámetros dinámicos

Funciones

Una función es un bloque reutilizable de código o declaraciones de programación diseñadas para realizar una determinada tarea. Para definir o declarar una función, Python proporciona la palabra clave def. La siguiente es la sintaxis para definir una función. El bloque de función de código se ejecuta solo si se llama o invoca la función.

Objetivo 1: Aprender a usar funciones sin parámetros

```
#Creamos la función
def mostrar_bienvenida():
    print("Bienvenidos a nuestra red social")



#llamamos a la función
mostrar_bienvenida()
```

Objetivo 2: Aprender a usar funciones con parámetros fijos

```
def calcular_precio_bs(pre_dol,tas_cam):
    pre_bs=float(pre_dol)*float(tas_cam)
    return pre_bs


print("Por favor indicar el precio del curso en Dólares: ")
precio=input()

print("Por favor indicar la tasa de cambio: ")
tasa=input()

tot_bs=calcular_precio_bs(precio,tasa)

print("Por favor hacer transferencia o pago móvil de:
",tot_bs,"Bs")
```

Objetivo 3: Aprender a usar funciones con parámetros dinámicos

```
def mostrar_paises_clasificados(confereracion,*paises):
     for pai in paises:
        print("Pais clasificado el mundial: ",pai,"de
",confereracion)


mostrar_paises_clasificados("América del
Sur","Brasil","Argentina","Ecuador","Uruguay")


mostrar_paises_clasificados("Europa","Serbia","Dinamarca","
España", "Países
Bajos","Suiza","Croacia","Francia","Inglaterra","Bélgica","
Alemania", "Portugal", "Polonia", "Gales")
```

Fin de Cuarto Ejercicio
Ejecutamos Commit

- `git add ejercicio4_funciones.py`
- `git commit -m “Ejercicio 4 – Python”`
- `git status`
- `git tag Ejercicio4`
- `git reflog`

POO – Clases y Objetos

- Objetivo 1: Aprender a definir clases
- Objetivo 2: Crear Constructores
- Objetivo 3: Instanciar objetos

Objetivo 1: *Aprender a Definir Clases*

En Python las clases se definen mediante la palabra clave class seguida del nombre de la clase, dos puntos (:) y a continuación, indentado, el cuerpo de la clase.

```
class pais:
```

Objetivo 2: Crear Constructores

El método __init__, con una doble barra baja al principio y final del nombre, se ejecuta justo después de crear un nuevo objeto a partir de la clase, proceso que se conoce con el nombre de instanciación. El método __init__ sirve, como sugiere su nombre, para realizar cualquier proceso de inicialización que sea necesario

```
def__init__(self,cod_pai,nom_pai,des_pai,ali_pai,
cti_pai,fky_con,est_pai):
      self.cod_pai=cod_pai
      self.nom_pai=nom_pai
      self.des_pai=des_pai
      self.ali_pai=ali_pai
      self.cti_pai=cti_pai
      self.fky_con=fky_con
      self.est_pai=est_pai
```

Objetivo 3: Instanciar objetos

```
#Llamamos a una función con el mismo nombre de la clase y podemos enviar los parámetros usados por el constructor.

#Estamos instanciando un objeto llamado p de la clase país
p=pais(1,"República Bolivariana de Venezuela","""República ubicada en Suramérica con mas de 30 millones de habitantes""","Vzla","+58",1,"A")

#Aquí estamos llamando a una función llamada agregar_país() que se encuentra dentro de la clase
p.agregar_pais()
```

Fin de Quinto Ejercicio
Ejecutamos Commit

- `git add ejercicio5_poo.py`
- `git commit -m “Ejercicio 5 – Python”`
- `git status`
- `git tag Ejercicio5`
- `git reflog`

POO – Clases y Objetos

• Objetivo 1: Comprender y aprender a usar la Herencia simple en Python

Objetivo 1: *Herencia Simple*

En un lenguaje orientado a objetos cuando hacemos que una clase (subclase) herede de otra clase (superclase) estamos haciendo que la subclase contenga todos los atributos y métodos que tenía la superclase. No obstante al acto de heredar de una clase también se le llama a menudo "extender una clase"

Objetivo 1: *Herencia Simple*

Para indicar que una clase hereda de otra se coloca el nombre de la clase de la que se hereda entre paréntesis después del nombre de la clase:

```
class conexion:
.....
class pais(conexion):
```

Opciones para usar funciones de una clase Padre

```
class conexion:

.....

class pais(conexion):


#Dentro de la clase hijo(pais)
#Opcion 1:
conexion.__init__(self) #constructor del padre
conexion.conectarse(self) #función del padre

#Opción2:
super().__init__()    #superclase padre (conexión)
super().conectarse()  #función del padre
```

Fin de Sexto Ejercicio
Ejecutamos Commit

- `git add ejercicio6_poo.py`
- `git commit -m “Ejercicio 6 – Python”`
- `git status`
- `git tag Ejercicio6`
- `git reflog`

POO – *Herencia Múltiple*

• Objetivo : Comprender y utilizar la Herencia Múltiple en Python

Objetivo 1: *Herencia Múltiple*

En Python, a diferencia de otros lenguajes como Java o C#, se permite la herencia múltiple, es decir, una clase puede heredar de varias clases a la vez.

Solo debemos agregar el nombre de las clases separadas por comas al momento de definir la clase.

Objetivo 1: *Herencia Múltiple*

En el caso de que alguna de las clases padre tuvieran métodos con el mismo nombre y número de parámetros las clases sobrescribirían la implementación de los métodos de las clases más a su derecha en la definición.

```
class publicacion(multimedia,usuario):
```

Objetivo 1: *Herencia Múltiple*

```
class publicacion(multimedia,usuario):
    def programar_publicacion(self):
        print("Esta función sirve para programar
una publicacion")

p=publicacion()     #instanciamos un objeto
p.agregar_usuario() #de la case usuario
p.agregar_foto()    #de la clase multimedia
p.imprimir()  #si hay una función imprimir en la
clase multimedia y otra en la clase usuario se
ejecuta la de la clase multimedia por estar
heredada como primer parámetro
```

Fin de Séptimo Ejercicio
Ejecutamos Commit

- `git add ejercicio7_poo.py`
- `git commit -m “Ejercicio 7 – Python”`
- `git status`
- `git tag Ejercicio7`
- `git reflog`

POO – Encapsulación

• Objetivo : Entender el concepto de Encapsulación y aplicarlo en Python

POO – Encapsulación

Encapsulación se refiere a impedir el acceso a determinados métodos y atributos de los objetos.

Esto se consigue en otros lenguajes de programación como Java utilizando modificadores de acceso que definen si cualquiera puede acceder a esa función o variable (public) o si está restringido el acceso a la propia clase (private).

POO – Encapsulación

En Python el acceso a una variable o función viene determinado por su nombre: si el nombre comienza con dos guiones bajos (y no termina también con dos guiones bajos) se trata de una variable o función privada.

```
def __datos_conexion(self): #Función Privada

self.__clave=12345 #Variable privada
```

POO – Encapsulación

```
class seguridad:
    def __init__(self):
        self.__usuario="pedro.parra82@gmail.com"
        self.__clave=12345
        self.nombre="Pedro Parra"

    def __datos_conexion(self):
        print("El usuario es: ",self.__usuario)
        print("La clave es: ",self.__clave)

    def mostrar_datos(self):
        self.__datos_conexion()

s=seguridad()
print("Si puedo observar el nombre del usuario porque es
público: ",s.nombre)

s.mostrar_datos()
```

Fin del Octavo Ejercicio
Ejecutamos Commit

- `git add ejercicio8_poo.py`
- `git commit -m “Ejercicio 8 – Python”`
- `git status`
- `git tag Ejercicio8`
- `git reflog`

POO

Temas por Ver Mas Adelante

- Polimorfismo
- Setter y Getters

POO – Módulos

- Objetivo : Entender el concepto de Módulos y aplicarlo en Python

Módulos

Los módulos son entidades que permiten una organización y división lógica de nuestro código.

Para importar un módulo se utiliza la palabra clave import seguida del nombre del módulo, que consiste en el nombre del archivo menos la extensión.

```
import ejercicio4_funciones
ejercicio4_funciones.mostrar_bienvenida()
```

Módulos

El import no solo hace que tengamos disponible todo lo definido dentro del módulo, sino que también ejecuta el código del módulo.

Fin del Noveno Ejercicio
Ejecutamos Commit

- `git add ejercicio9_poo.py`
- `git commit -m “Ejercicio 9 – Python”`
- `git status`
- `git tag Ejercicio9`
- `git reflog`

Enlace Python - Postgres

Psycopg2 es una **librería de Python que permite conectarse y trabajar con bases de datos PostgreSQL**. Con ella, se pueden ejecutar consultas SQL, realizar transacciones, manejar errores entre otras funciones.

Python Package Index (PIP)

pip es un sistema de gestión de paquetes utilizado para instalar y administrar paquetes de software escritos en Python. Muchos paquetes pueden ser encontrados en el Python Package Index.

Sintaxis:
pip install _____

Ejemplo:
pip install psycopg2

Instalación de Psycopg2 en Python

Paso 1: Ubicar la siguiente carpeta en nuestro computador.
(Habilita la visualización de carpetas ocultas.)

C:\Users_____\AppData\Local\Programs\Python\Python311\Scripts

Paso 2: Desde cmd como administrador nos vamos a la ruta del paso 1

Paso 3: pip install psycopg2

Paso 4: *Desde la consola de Python escribimos*
import psycopg2

Funciones mas usadas por psycopg2

Función	Descripción
connect()	Cree una nueva sesión de base de datos y devuelve un nuevo objeto de conexión.
cursor()	Permite que el código Python ejecute el comando PostgreSQL en una sesión de base de datos
execute()	Ejecutar una operación de base de datos (consulta o comando)
fetchall()	Obtenga todas las filas del resultado de una consulta y las devuelva como una ***lista de tuplas***. Se

Aprendiendo a usar psycopg2

```
import psycopg2
conexion = psycopg2.connect(user="postgres",
password="1234", host="localhost", port=5432,
database="diplomado")
cursor = conexion.cursor()

sql = "select * from ubicacion.continente"
cursor.execute(sql)
registro=cursor.fetchall()

for continente in registro:
    print (continente)

cursor.close()
conexion.close()
```

Ejercicios Propuestos

Usando psycopg2 hacer un CRUD de la tabla continente, una vez finalizado el ejercicio agregar los archivos a git.

Los nombres de los archivos serán:

- *ejercicio10_pg_select.py*
- *ejercicio11_pg_insert.py*
- *ejercicio12_pg_update.py*
- *ejercicio13_pg_delete.py*