Introducción a

Claudia Nallely Sánchez Gómez

Python, ¿qué es?

Python es un lenguaje de programación que permite trabajar rápidamente y su objetivo es tener una sintaxis simple para crear códigos legibles. Fue creado por Guido van Rossum y lanzado en 1991.

Puede ser usado para:

- Desarrollo web
- Conexión con base de datos
- Manejo de datos
- Etcétera

Más información en:

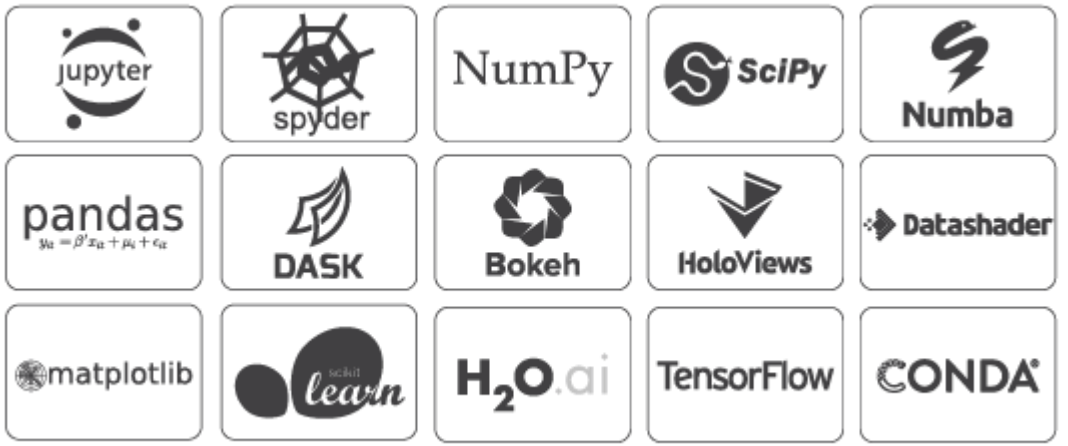
https://www.python.org/

Instalación

Se recomienda utilizar la distribución Anaconda, ya que es código abierto y contiene varias librerías para ciencia de datos y aprendizaje de máquina.

Más información en:

https://www.anaconda.com



Claudia Nallely Sánchez Gómez cnsanchez@up.edu.mx

UNIVERSIDAD PANAMERICANA®

Impresión

Programa hola_mundo.py

```
print("hola mundo")
```

Otras formas de hacerlo

```
print('hola mundo')
```

```
print('hola','mundo')
```

- En Python los archivos de código se guardan con la extensión .py
- Las cadenas de texto pueden estar representadas entre comillas dobles o entre comillas simples. Si tu cadena incluye comillas dobles, puedes utilizar comillas simples para definir el inicio y fin de la cadena, y viceversa.
- La función print puede recibir varios parámetros separados por coma

Comentarios

```
# comentario
print("hola 1")
print('hola','2') # comentario
'''
comentario en
múltiples líneas
'''
print('hola 3')
```

- # comenta todos los caracteres que siguen después de # en la misma línea
- Triple comilla sirve para comentar en varias líneas

Variables - cadenas

```
nombre = "Valeria"   # Asignación simple
nombre1, nombre2 = "Luis", 'Cindy' # Asignación múltiple a múltiples variables
apellido1 = apellido2 = "Sanchez" # Asignación del mismo valor

print(nombre)
print(nombre1,nombre2)
print(apellido1,apellido2)
```

- En Python no tiene un comando para declarar variables, estás se crean en el momento en que les asignas un valor.
- La asignación de valores puede ser simple o múltiple, además de que se puede asignar el mismo valor a varias variables en una línea

Variables - cadenas

- Para marcar el inicio y fin de las cadenas se pueden utilizar comillas simples o dobles.

```
a,b = 'hola "buen" día',"adiós"
print(a,b)
```

- También se pueden asignar cadenas de varias líneas con triple comilla.

```
a = ''' Cadena
de varias
lineas '''

b = """
Cadena
con dos lineas
"""
print(a,b)
```

Variables - números

```
n = 'Clau' # str
x = 6      # int
y = 7.2    # float
z = 3+2j   # complex

print(n,type(n))
print(x,type(x))
print(y,type(y))
print(z,type(z))
```

- Los tipos de datos numéricos son: int, float y complex
- La función type regresa el tipo de variable

Variables - números

- Python distingue entre int y float por el número decimal

```
x = 4  # int
y = 4. # float
print(x,type(x))
print(y,type(y))
```

- Para asignar flotantes se pueden utilizar varias notaciones

```
x = 35e3       # float
y = 12E4       # float
z = -87.7e100  # float
print(x,type(x))
print(y,type(y))
print(z,type(z))
```

Variables - booleanas

- Las variables booleanas toman los valores True y False

```
x = True
y = False
print(x,type(x))    # True bool
print(y,type(y))    # False bool
```

Variables - cast

- Las funciones cast sirven para cambiar el tipo de variable

```
a = int(1)           # x es 1
b = int(2.8)         # y es 2
c = int("3")         # z es 3
d = int("3.3")       # ERROR
e = int(3+2j)        # ERROR
error = int("hola") # ERROR
```

```
a = float(1)           # x es 1.
b = float(2.8)         # y es 2.8
c = float("3")         # z es 3.
d = float("3.3")       # z es 3.3
e = float(3+2j)        # ERROR
error = float("hola") # ERROR
```

```
a = str(1)
b = str(2.8)
c = str(3+2j)
d = str("hola")
```

Strings

- Como en otros lenguajes de programación, las cadenas pueden verse como arreglos de caracteres.
- Se pueden acceder a posiciones negativas, siendo -1 la última posición del arreglo, -2 la penúltima y así sucesivamente.
- También se pueden obtener una subcadena con la sintaxis cadena[idx1: idx2], la subcadena incluirá el carácter en el índice idx1 pero no el idx2.
- También se pueden obtener los primeros k caracteres cadena[: k]
- O los últimos k cadena[-k :]

```
a = "cadena de texto"
print(a[0],a[2]) # c d
print(a[-1],a[-2]) # o t
print(a[1:3],a[-4:-1]) # ad ext
print(a[:4],a[-3:]) # cade xto
```

Strings

¿Qué imprime el siguiente código?

```
s = 'aguascalientes'
print(s[0],s[1:4],s[-1:])
```

Strings

- El método len() regresa la longitud de una cadena
- El método lower() regresa la cadena en minúsculas
- El método upper() regresa la cadena en mayúsculas
- El método replace() reemplaza una cadena por otra
- El método split() divide una cadena en varias utilizando un carácter en específico

```
s = "Cadena de Texto"
print(len(s))              # 15
print(s.lower())           # cadena de texto
print(s.upper())           # CADENA DE TEXTO
print(s.replace('a','x')) # Cxdenx de Texto
print(s.split(' '))        # ['Cadena', 'de', 'Texto']
```

Operadores

Aritméticos

Operador	Nombre
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo
**	Exponente
//	División entera
operador =	Asignación + operación

Relacionales

Operador	Nombre
==	Igual
!=	No igual
>	Menor
<	Mayor
>=	Menor o igual
<=	Mayor o igual

Lógicos

Operador
and
or
not

Listas

List es una lista de elementos que se pueden editar y pueden ser de diferentes tipos.

- Se puede acceder a los elementos con el operador corchete []
- El método len() devuelve el número de elementos en la lista

```
lista = ['hola',4,6.7,'a']
print(lista)      # ['hola', 4, 6.7, 'a']
print(lista[-1])  # a
print(lista[:2])  # ['hola', 4]
print(len(lista)) # 4
```

Listas

Función	Descripción
append()	Agrega un elemento al final de la lista
clear()	Elimina todos los elementos de la lista
copy()	Regresa una copia de la lista
reverse()	Invierte el orden de la lista
sort()	Ordena la lista, si son tipos diferentes marca error
extend()	Agrega los elementos de una lista al final de la lista actual
count()	Regresa el número de elementos con un valor específico
index()	Regresa el índice del primer elemento con un valor específico
remove()	Elimina un elemento con un valor específico
insert()	Agrega un elemento en una posición específica
pop()	Elimina un elemento en una posición específica, si no se envía la posición elimina el último elemento

Listas

¿Qué imprime el siguiente código?

```
lista = ['a','z','m','e']
lista.append('c')
lista.insert(2,'b')
lista.reverse()
lista.sort()
lista.remove('a')
lista.pop()
print(lista)
```

Indentación

A diferencia de otros lenguajes de programación, Python utiliza indentación para definir instrucciones anidadas. Se utilizan 4 espacios en blancos para definir una instrucción dentro de otra.

```
def factorial(n):
    f = 1
    for i in range(1,n+1):
        f*= i
    return f

print(factorial(4))
```

If...else

if [condición] :
    [instrucciones]
elif [condición] :   # opcional
    [instrucciones]
else: # opcional
    [instrucciones]

```
calif = float(input("¿Cuál es tu calificación?:"))
if calif >= 9. :
    print('Excelente')
elif calif >= 8. :
    print('Bien')
elif calif >= 6. :
    print('Aprobado')
else:
    print('Reprobado')
```

If…else en una línea

```
calif = float(input("¿Cuál es tu calificación?:"))

mensaje = 'excelente' if calif>= 9.0 else 'bien'
print(mensaje)
```

```
calif = float(input("¿Cuál es tu calificación?:"))

mensaje = 'excelente' if calif>= 9.0 else 'aprobado' if calif>=6.0 else 'reprobado'
print(mensaje)
```

while

while [condición] :
    [instrucciones]

¿Qué imprime el siguiente código?

```
i,j = 1,10
while i<j and j>2:
    i+=1
    j-=1
    if j==9:
        continue
    if i == 5:
        break
    print(i,j)
```

for

En Python, la sintaxis for es como la sintaxis for each de otros lenguajes.

for [elemento] in [objeto iterable] :
    [instrucciones]

```
lista = ['a','x','m','y']
for elemento in lista:
    print(elemento)

for e in 'aguascalientes':
    print(e)
```

for – range()

La función range nos permite crear una lista numérica iterable para ejecutar en for.

range(idxInicio, idxFin, incremento)

- idxInicio, incremento son opcionales.
- Por default, los valores de idxInicio e incremento son 0 y 1 respectivamente.

```
# 0 1 2 3 4 5 6 7 8 9
for i in range(10):
    print(i)

# 5 6 7 8 9
for i in range(5,10):
    print(i)

# 2 4 6 8
for i in range(2,10,2):
    print(i)
```

for – enumerate(), zip()

- enumerate() itera una lista devolviendo el índice y el elemento
- zip() itera sobre los elementos dos o más listas

```
lista = ['a','b','c','d']
listaN = [10,20,30,40]

for index,elemento in enumerate(lista):
    print(index,elemento)

for n,e in zip(listaN,lista):
    print(n,e)
```

Ejercicios

1. Escribe un programa que pida al usuario un número y que imprima su tabla de multiplicar.

2. Escribe un número que pida al usuario un número y verifique si contiene el dígito 5.

3. Escribe un programa que reciba una frase y diga si es palíndroma o no. Se deben quitar los espacios y validar minúsculas y mayúsculas por igual. Ejemplo: “Anita lava la tina”, “Oxxo”.

4. Escribe una función que tome dos listas y cree una nueva alternando los elementos de las listas.
   Por ejemplo: [a,b,c], [1,2,3] → [a,1,b,2,c,3].

5. Escribe un programa que lea una contraseña y diga si es válida o no.
   Las contraseñas válidas deben:
   - Tener al menos 8 caracteres
   - Al menos una mayúscula, una minúscula y un carácter especial

Claudia Nallely Sánchez Gómez cnsanchez@up.edu.mx

Listas, tuplas, conjuntos, diccionarios

	Descripción	¿Se pueden duplicar los elementos?	¿Se puede editar?	Carácter con el que se representa
Lista	Es una lista de elementos	Si	Si	[ ]
Tupla	Es una lista de elementos	Si	No	( )
Conjunto	Representa un conjunto de elementos	No	Si	{ }
Diccionario	Representa elementos siguiendo la estructura clave-valor	No	Si	{ }

Tuplas

```
lista = ['a','b','c']
tupla = ('a','b','c')

print(lista,tupla)
# ['a', 'b', 'c'] ('a', 'b', 'c')

lista[0]='A'
tupla[0]='A' # ERROR
```

Se puede utilizar las siguientes funciones

Función	Descripción
count()	Regresa el número de elementos con un valor específico
index()	Regresa el índice del primer elemento con un valor específico

Conjuntos

Son conjuntos de elementos sin duplicaciones.

```
s = {'a','c','z'}
s.add('a')
s.add('b')
print(s)
# {'c', 'b', 'z', 'a'}

s2 = set()
s2.add(1)
s2.add('hola')
s2.add(2)
s2.add('hola')
print(s2)
# {1, 2, 'hola'}
```

Conjuntos

Función	Descripción
add()	Agrega un elemento
clear()	Elimina todos los elementos del conjunto
copy()	Regresa una copia del conjunto
difference()	Regresa un conjunto que contiene la diferencia entre dos conjuntos
difference_update()	Elimina los elementos en el conjunto actual que están incluidos en otro conjunto
discard()	Elimina el elemento específicado
intersection()	Regresa un set que es la intersección de dos conjuntos
intersection_update()	Elimina los elementos en el conjunto actual que NO están incluidos en otro conjunto
isdisjoint()	Regresa si dos conjuntos tienen una intersección o no
issubset()	Regresa si otro conjunto contiene al conjunto actual o no
issuperset()	Regresa si este conjunto contiene al otro conjunto o no
pop()	Elimina un elemento del cojunto
remove()	Elimina un elemento específico del conjunto
symmetric_difference()	Regresa un conjunto con la diferencia simétrica entre dos conjuntos
symmetric_difference_update()	Inserta la diferencia simétrica de este conjunto y otro
union()	Regresa un conjunto que contiene la unión entre dos conjuntos
update()	Actualiza el conjunto actual con la unión del conjunto y otro

Diccionarios

Es una estructura de datos que permite el almacenamiento y búsqueda eficiente de información tipo clave valor.

```
s = dict()
s['a']='aula'
s['b']='balon'
print(s)
# {'a':'aula', 'b': 'balon'}
```

```
s = {1:'uva',3:'mango',10:'fresa'}
s[6]='naranja'
s[1]='melon'

print(s)
for k,v,t in zip(s.keys(),s.values(),s.items()):
  print(k,v,t)
#{1: 'melon', 3: 'mango', 10: 'fresa', 6: 'naranja'}
#1 melon (1, 'melon')
#3 mango (3, 'mango')
#10 fresa (10, 'fresa')
#6 naranja (6, 'naranja')
```

Diccionarios

Función	Descripción
clear()	Elimina todos los elementos de un diccionario
copy()	Regresa una copia del diccionario
fromkeys()	Regresa un diccionario con las llaves especificadas
get()	Regresa el valor de una llave específica
items()	Regresa una lista que contiene una tupla por cada par llave-valor
keys()	Regresa una lista con las llaves
pop()	Elimina el elemento de una llave específica
popitem()	Elimina el último elemento insertado
values()	Regresa una lista con los valores

Funciones

En Python, la sintaxis de una función es:

def nombre_funcion([parámetros separados por coma]):
  [instrucciones]

```
def suma(n1,n2):
    return n1 + n2

r = suma(4,5)
print("El resultado es:",r)
```

Una gran ventaja es que se pueden devolver más de dos valores

```
def suma_multiplicacion(n1,n2):
    suma = n1+n2
    mult = n1*n2
    return suma, mult

a = 4
b = 5
s,m = suma_multiplicacion(a,b)
print(s,m)
```

Funciones

Los parámetros de las funciones se pasan por copia si son tipo primitivo y por referencia si son alguna estructura tipo lista, tupla, conjunto o diccionario.

```
def funcion(vint,vfloat,vstr,vbool,vlist):
    vint +=10
    vfloat +=10
    vstr = 'modificado'
    vbool = False
    vlist[0] = 'modificado'

vint = 4
vfloat = 4.4
vstr = 'cadena'
vbool = True
vlist = [1,2,3,4]
print(vint,vfloat,vstr,vbool,vlist)   # 4 4.4 cadena True [1, 2, 3, 4]
funcion(vint,vfloat,vstr,vbool,vlist)
print(vint,vfloat,vstr,vbool,vlist)   # 4 4.4 cadena True ['modificado', 2, 3, 4]
```

Funciones

Las funciones permiten valores por default

```
def funcion(a = 1, b = 2):
    print('a',a,'b',b)

funcion()
# a 1 b 2

funcion(10,20)
# a 10 b 20

funcion(10) # Los valores envíados se asignan de izquierda a derecha
# a 10 b 2

funcion(b=20) # Se pueden asignar valores en diferente orden indicando la variable
# a 1 b 20
```

Clases

Las clases tienen las siguientes características:

- El uso de self (this en otros lenguajes de programación) es obligatorio, como primer parámetro de los métodos y para usar las variables de clase.
- El constructor es __init__, se recomienda inicializar ahí todas las variables
- Para indicar que una variable o método es privado se utiliza un doble guion bajo al inicio del nombre
- __str__ es el método que se sobre escribe para poder imprimir un objeto con la función print

```
class Persona:
    def __init__(self,nombre='',edad=0):
        self.__nombre = nombre
        self.__edad = edad

    @property
    def nombre(self):
        return self.__nombre
    @nombre.setter
    def nombre(self,v):
        self.__nombre = v

    @property
    def edad(self):
        return self.__edad
    @edad.setter
    def edad(self,v):
        self.__edad = v if v>=0 else -v

    def __str__(self):
        return self.__nombre +' ' +str(self.__edad)
```

Clases

```
class Persona:
    def __init__(self,nombre='',edad=0):
        self.__nombre = nombre
        self.__edad = edad

    @property
    def nombre(self):
        return self.__nombre
    @nombre.setter
    def nombre(self,v):
        self.__nombre = v

    @property
    def edad(self):
        return self.__edad
    @edad.setter
    def edad(self,v):
        self.__edad = v if v>=0 else -v

    def __str__(self):
        return self.__nombre +' ' +str(self.__edad)
```

```
p1 = Persona('Valeria',19)
p2 = Persona()
p2.nombre = 'Ernesto'
p2.edad = -15

print('Persona 1:',p1)
#Persona 1: Valeria 19

print('Persona 2:',p2)
#Persona 2: Ernesto 15
```

Herencia

```
class Empleado(Persona):
    def __init__(self,nombre='',edad=0,salario=0):
        Persona.__init__(self,nombre,edad)
        self.__salario = 100

    @property
    def salario(self):
        return self.__salario

    @salario.setter
    def salario(self,v):
        self.__salario = v if v>=0 else -v

    def __str__(self):
        return Persona.__str__(self)+' '+str(self.__salario)

e = Empleado('Ana',20,300)
print(e)
```

Ejercicio

Crea un programa que en base a la siguiente tupla:

```
tupla = ( ('Aguascalientes', 'mujer'),
        ('Zacatecas', 'hombre'),
        ('Guanajuato', 'mujer'),
        ('Zacatecas', 'mujer'),
        ('Durango', 'hombre'),
        ('Aguascalientes', 'hombre'),
        ('Zacatecas', 'hombre'),
        ('Jalisco','hombre'),
        ('Zacatecas', 'hombre'),
        ('Zacatecas', 'mujer'))
```

Imprima los diferentes estados de la republica además del número de hombres y mujeres en cada estado.
Utilice diccionarios y/o conjuntos.

Ejercicio

Crear la clase Figura con las siguientes características:

- Atributos privados: nombre y color. Crear los métodos para editar y regresar los valores.
- Crear un constructor donde se pidan los dos atributos.
- Sobre escribir el método __str__ para imprimir la información de la figura.
- Crear los métodos vacios calcular_perimetro y calcular_area

Crear la clase Rectangulo que herede de Figura con las siguientes características:

- Atributos privados: base y altura. Crear los métodos para editar y regresar los valores, validando que los valores sean positivos.
- Crear un constructor que pida los atributos nombre, color, base y altura. Por default el valor de base y altura es 1.
- Sobre escribir el método __str__ para imprimir la información de la figura.
- Implementar los métodos calcular_perímetro y calcular_area

Crear la clase Cuadrado que herede de Figura.