TALLER DE
PROGRAMACIÓN EN PYTHON: PROGRAMA DE FORMACIÓN DE FORMADORES TI

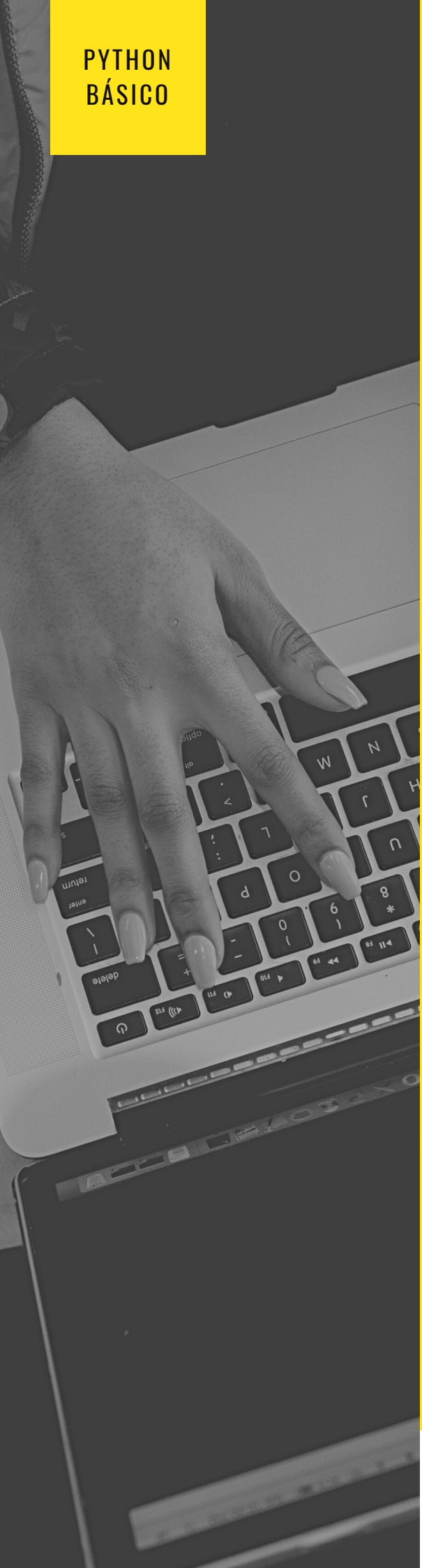
Python Básico

REDIS
RED COLOMBIANA DE PROGRAMAS DE INGENIERÍA DE SISTEMAS Y AFINES

Fedesoft

UNAD
Universidad Nacional Abierta y a Distancia

PYTHON
BÁSICO



CONFERENCISTAS

ARELLYS CORREA

Ingeniera de Sistemas
Docente UNAD

@arellyscorrea

KEVIN MARTINEZ

Ingeniero de Sistemas

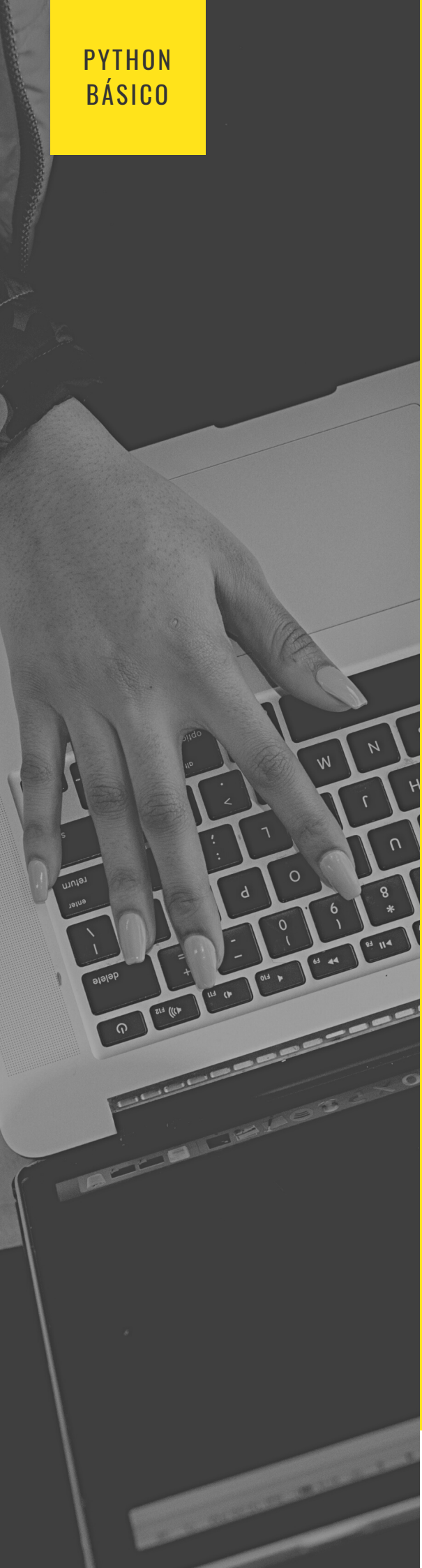
@keii_25

REDIS
RED COLOMBIANA DE PROGRAMAS DE INGENIERÍA DE SISTEMAS Y AFINES

Fedesoft

UNAD
Universidad Nacional
Abierta y a Distancia

PYTHON
BÁSICO



CONTENIDO

REDIS
RED COLOMBIANA DE PROGRAMAS DE INGENIERÍA DE SISTEMAS Y AFINES
Fedesoft
UNAD
Universidad Nacional
Abierta y a Distancia

¿QUÉ ES PYTHON?

Lenguaje de programación interpretado de alto nivel. (POO, funcional, imperativa)

- Creado en los años 90 por Guido van Rossum
- Su nombre procede del programa de la BBC “Monty Python’s Flying Circus”
- Distribuciones de código abierto
- Amplia biblioteca estándar
- Curva de aprendizaje baja

RAZONES PARA APRENDER PYTHON

- Proyecciones Futuras: Idóneas
- Python, está creciendo entre los desarrolladores y prueba de ella son las recientes encuestas realizadas por la comunidad de Stack Overflow que lo sitúan entre el más deseado.
- Multiplataforma
- Lenguaje de programación interpretado: Funciona en cualquier tipo de sistema que integre su interpretador.
- Por defecto: GNU/Linux & MacOS
- Libre: Código Abierto
- Contribución al desarrollo y divulgación del mismo.
- Administrado por Python Software Foundation, una organización sin ánimo de lucro.
- Lenguaje productivo: Potente, flexible y sintaxis clara.

VALORACIÓN ACTUAL

Empresas como: Google, Walt Disney, la NASA, Yahoo! Facebook o YouTube Nokia utilizan Python para desarrollar sus productos y servicios

VARIABLES Y TIPOS DE DATOS

Las variables permiten almacenar en un espacio de memoria del programa: datos.

Estos datos serán de un tipo u otro, acorde o en función de la información que se guarde en ellas.

```
1 nombre = 'Juan'  # cadena de texto o string
2 apellido = 'Lopez'
3 edad = 27 # número entero
```

CONSIDERACIONES

- Comenzar con una letra o guion bajo
- El cuerpo del nombre de la variable está formado
- por letras, números o guiones bajos.
- Los nombres de las variables son de tipo case sensitive, es decir, que no sera lo mismo que una variable se llame *edad* a *EDAD*.
- Por último, hay una serie de palabras que no
- pueden utilizarse (def, global, return, if, for...)

Ejemplos

```
1 _nombre = 'Maria'  # cadena de texto o string
2 nombre = 'Pedro'
3 NOMBRE = 'Pedro' # número entero
```

Tipos de Variables

```
1 nombre = 'Maria'  # cadena de texto o string
2 edad = 'Pedro'
3 título_libro = 'Aprendiendo Python'
4 estado = True # booleano
5 precio = 25.8 # número flotante (float)
```

Lectura de datos en python: INPUT()

La funcion input() permite intruducir datos por teclado

```
1 nombre = input()
2
3 Juan
4
5 print(nombre)
>> Juan
```

En el siguiente ejemplo, podemos enviar un mensaje al usuario antes de ingresar los datos.

```
1 nombre = input('Escribe tu nombre: ')
2
3 Escribe tu nombre:Juan
4
5 print(nombre)
>> Juan
```

NUMEROS

Python soporta dos tipos de números: Enteros (integer) y de punto flotante (float)

```
1  # integer
2
3  x = 10
4  print(x)
5  >> 10
6
7  # float
8
9  y = 3.8
10 print(y)
11 >> 3.8
```

FUNCIÓN *TYPE()*

Con la funcion type, mostramos el tipo de variable que es.

```
1  x = 20
2
3  >> type(x)
4  <class 'integer'>
5
6  y = 3.9
7  >> type(y)
8  <class 'float'>
```

Cadenas de Texto (string)

Las cadenas de texto o string se definen mediante comillas simple (' ') o comillas dobles (" ").

```
name_one = 'Maria'

name_two : "Pedro"

post = 'Libro: "Cien Años de Soledad"'

frase = """Esto
             es una cadena
                larga """
```

Concatenación de string

Unir string con el operador +

```
primera_palabra = 'Hola'

segunda_palabra = " Mundo"

frase_completa = primera_palabra +  segunda_palabra

print(frase_completa)
>> Hola Mundo
```

Metodo alternativo: str.format()

```
# Ordenar por defecto

frase = "Meses: {}, {} y {}.format('Enero', 'Febrero', 'Marzo')
print(frase)

# Ordenar especificando la posición

frase = "Meses: {1}, {0} y {2}".format('Enero', 'Febrero', 'Marzo')
print(frase)

# Ordenar especificando la posición

frase = "Meses: {ene}, {feb} y {mar}.format(ene='Enero', feb ='Febrero', mar='Marzo')
print(frase)
```

Cadenas 'f' (f-strings)

Los llamados f-string es una forma comoda y directa de insertar variables y expresiones de una cadena de texto.

```
1  nombre = 'Maria'
2
3  edad = 27
4
5  print(f'Me llamo {nombre} y tengo {edad} años')
6
```

Conversion de Tipos

Al momento de concatenar un string con varibales como integer o float puede habler problemas

```
1  nombre = 'Maria'
2
3  edad = 27
4
5  print('Soy' + nombre + ' y tengo ' + edad + ' Años')
6  # ERROR
```

Mediante la funcion str() podemos convertir un valor a string y evitar porblemas de concatenación.

```
1  nombre = 'Maria'
2
3  edad = 27
4
5  print('Soy' + nombre + ' y tengo ' + str(edad) + ' Años')
6
```

Metodos en cadenas de texto (string)

En python es posible obtener un ca carácter concreto-especifico de uns string utilizando corchetes [ ] y el índice del carácter al que queremos acceder

```
1  post= 'Aprendiendo programación con Python'
2
3  post[0] # devuelve el primer caracter
4
5  post[1] # devuelve el segundo caracter
6
7  post[-1] # devuelve el primer caracterempezando por el final
```

Obteniendo un substring

```
1  post= 'Aprendiendo programación con Python'
2  # devuelve los caracteres desde la posición 0 hasta la 4 (No incluye la posición 4)
3  post[0-4]
4  # En caso de dejar la primera variable varia, se considera la primera posición
5  post[:4] # devuelve el segundo caracter
6
7
```

Otros métodos utiles

```
1  len(str)       # Devuelve la longitud del string
2
3  str.upper()    # Convierte a MAYUSCULAS
4
5  str.lower()    # Convierte a minusculas
6
7  str.title()    # Convierte a Mayúsculas la primera letra de cada palabra
8
```

OPERADORES Y EXPRESIONES

Los operadores son simbolos especiales de python, que permiten realizar operaciones aritméticas o lógicas

Operador	Ejemplo	Significado
+	a + b	Suma
-	a - b	Resta
-	-a	Negacion(asignar valor negativo
*	a * b	Multiplicación
/	a / b	División
%	a % b	Módulo (resto de la division)
//	a // b	División entera
**	a ** b	Exponente

EJEMPLOS

```
1  x = 5 # Creamos una variable de tipo integer
2
3  y = 2 # Creamos otra variable de tipo integer
4
5  print(x + y) # 7
6  print(x - y) # 3
7  print(x * y) # 10
8  print(x / y) # 2.5
9
```

OPERADORES RELACIONALES O DE COMPARACIÓN

Operador	Ejemplo	Significado
>	a > b	Mayor que: True si a es mayor que b
<	a < b	Menor que: True si a es menor que b
==	a == b	Igual: True si a y b son iguales
!=	a!= b	Distinto: True si a y b son distintos
>=	a >= b	Mayor o igual: True si a es igual o mayor que b
<=	a <= b	Menor o igual: True si a es igual o menor que b

OPERADORES LÓGICOS

Evaluan valores devolviendo tambien True o False como resultado

Operador	Ejemplo	Significado
and	a **and** b	**True** si a y b son **True**
or	a **or** b	**True** si a o b son **true**
not	**not** b	**True** si b es falso

ESTRUCTURA DE CONTROL

Condicionales

Las estructura de control se utilizan para ejecutar bloques de código en función de condiciones

Sentencia IF - ELSE

Se evalúa la condición especificada en la sentencia **if** y en caso de que se cumpla, se ejecutara el bloque de codigo indentado (tabulado). En caso de que el resultado de la condición sea **False**, el bloque especificado no se ejecutara

```
1  numero = 10
2
3  if numero > 1:
4     # Este bloque de codigo se ejecuta cuando la condicion sea True
5     print("Es mayor que uno")
7
```

Otros ejemplos

```
1  edad = 18
2  altura = 165
3  if (edad >= 18 and altura >= 165):
4     # Este bloque de codigo se ejecuta cuando se cumplan las dos condiciones
5     print('Puede ocupar asiento en la Montaña Rusa')
6
```

Mediante la palabra reservada else es posible efectuar un bloque de codigo que se ejecute en caso de que la condición inicial no se cumpla.

```
1  edad = 18
2  altura = 165
3  if (edad >= 18 and altura >= 165):
4     # Este bloque de codigo se ejecuta cuando se cumplan las dos condiciones
5     print("Puede ocupar asiento en la Montaña Rusa")
6  else:
7     # Este bloque de codigo se ejecuta cuando no se cumplan una o ninguna condición
8     print("No puede pasar a la Montaña Rusa")
```

Tambien podemos comprobar más condicionales mediante al expresion elif. En este caso, se seguiran comprobando todas las condiciones elif que una de ellas se cumpla. En caso contrario, se ejecutara el bloque de código dentro del else (en el caso que lo hubiera)

```
numero = 10

if numero < 5:
   print("Es menor que 5")
elif numero < 8:
   print("El numero esta entre el 5 y el 7")
else:
   print("El numero es mayor o igual 10")
```

Condicionales: anidadas

```
numero = 4

if numero >= 0 :
   if numero == 0:
       print('El valor es 0')
   else:
       print("Es un numero positivo")
else:
   print("Es un numero negativo")
```

Bucles

Los bucles permiten ejecutar un bloque de codigo tantas veces como se quiera.

Sentencia WHILE

Permite ejecutar un bloque de código mientras la expresión que definamos se cumpla (Es decir, devuelva True). Python interpretará como True cualquier valor distinto a 0 o None.

```
1  contador = 0
2
3  while(contador < 5):
4     # Se ejecutara mientras la variable contador sea menor a 5
5     contador = contador + 1
6     print("Interación número: ", contador)
7
8  print("Ejecucion finalizada")
9
```

Sentencia: Break

Para detener una ejecución de forma voluntaria se utiliza la sentencia break

```
1  contador = 0
2
3  while(contador < 5):
4
5     contador = contador + 1
6     print("Interación número: ", contador)
7     if contador == 3:89     break
8  print("Ejecucion finalizada")
9
```

Sentencia: continue

Es posible saltar unicamente la iteraccion actual mediante la sentencia continue

```
1  contador = 0
2
3  while(contador < 5):
4
5     contador = contador + 1
6
7     if contador == 3:
8       continue
9     print("Interación número:{} ".format(contador))
10 print("Ejecucion finalizada")
```

Bucle WHILE - ELSE

La expresión else puede utilizarse conjuntamente tras un bloque while. De esta forma se puede definir un bloque de codigo que se ejecutará una vez finalizado el bloque while.

```
1  contador = 0
2
3  while(contador < 5):
4
5     contador = contador + 1
6     print("Interación número:{} ".format(contador))
7
8  else:
9     print("Ejecucion finalizada")
```

Sentencia FOR

A diferencia de otros lenguajes de programación, python utiliza la sentencia FOR para iterar unicamente por secuencias (listas, tuplas, cadenas de caracteres...)

```
1  frutas = ["Manzana", "Pera", "Piña", "Uvas"]
2
3  for f in frutas:
4     print(f)
5
```

Sentencia FOR

```
1  post = ["Aprendiendo Python"]
2
3  for c in post:
4     print(c)
5
```

Deteniendo la ejecucion con break

```
1  numeros = [2, 5, 7, 1, 8, 3]
2  total = 0
3  for n in numeros:
4     total += 1
5     if total > 10:
6        break
```

Bucle FOR - ELSE

Python permite definir un bloque de código que se ejecutará una vez finalice la iteracción por todos los elementos de la lista. Importante: No se ejecutara si se ha finalizado mediante break

```
1  estudiantes = ['Juan', 'Felipe', 'Maria', 'Luisa',  'Pedro']
2
3  for estudiante in estudiantes:
4     print('Nombre: ', estudiante)
5  else:
6     print('Lista Terminada')
```

Función range()

La función range([start,] stop [,step]) devuelven una secuencia de números. Es por ello que se utiliza de forma frecuente apra iterar.

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

iteraccion de lista utilizando el índice

```
estudiantes = ['Luis', 'Sandra', "Julio"]

for i in range(len(estudiantes)):
    print(estudiantes[i])


```

Listas y Tuplas

Las listas permiten guardar más de un elemento dentro de una variable, y ademas hacerlos en un orden concreto.
Pueden contener un número ilimitado de elementos de cualquier tipo.

```
# Lista vacia

lista_vacia = []

# Lista con valores

elementos = ["Piña", "Naranja", "Manzana", 3, 3.6, [1, 5, 7, "Maria"]]


# Acceder a Elementos de la lista

print(elementos[0]) # Muestra Piña
print(elementos[2]) # Muestra Manzana
print(elementos[5]) # Muestra la lista ultima creada
elementos[0] = 'Uva' # Cambiar un elemento en la posición 0

```

Métodos mas utilizados con las listas

Método	Acción
frutas.***append***("Pera")	Inserta "Pera" al final de la lista
frutas.***insert***(**1**, "Limon")	Inserta "Limon" en la posición 1
frutas.***remove***("Pera")	Elimina la primera aparición de "Pera" de la lista
frutas.***pop***()	Elimina el ultimo elemento de la lista
frutas.***pop***(5)	Elimina el sexto elemento de la lista
frutas.***clear***()	Elimina todos los elementos de la lista
frutas.***index***("Pera")	Devuelve el indice de la primera aparición de "Pera"
frutas.***sort***()	Ordena la lista (Estos elementos deben ser comprables)
frutas.***reverse***()	Ordena la lista en orden inverso
frutas.***copy***()	Devuelve una copia de la lista
frutas.***extend***(frutas_2)	Fusiona las dos listas

Tuplas

Las tuplas son listas inmutables; esto quiere decir, que una vez declaradas o definidas, no se pueden realizar acciones modificables sobre ellas (añadir, eliminar o hacer modificaciones sobre los elementos)

```
numeros = (1,3,5,7,9)
print(numeros)   # salida (1,3,5,7,9)

# Acceder a las tuplas

numeros[1] # Resultado = 3
frutas = ("Banano", "Limon", "Pera")
fruta_1, fruta_2, fruta_3 = frutas    # unpack  ---> Desempaquetar

print(fruta_1) #  Banano
print(fruta_2) # Limon
print(fruta_3) # Pera

```

Diccionarios

Un diccionario, es un conjunto de pareja clave-valor (key-value)
Acceder a cada elemento por su clave.

```
persona = {
        "nombre" : "Juan",
        "apellido" : "Lopez",
        "edad": 25,
        "estatura" : 1.75,
        "estado_vinculacion-salud" : True
}

```

Acciones sobre un Diccionario

```
persona = {
        "nombre" : "Juan",
        "apellido" : "Lopez",
        "edad": 25,
        "estatura" : 1.75,
        "estado_vinculacion-salud" : True
}
edad = persona["edad"] # nos devuelve el valor de edad
estatura = persona.get("estatura") # nos devuelve el valor de estatura
persona["edad"] = 30
persona["estado_vinculacion_salud"] = False
persona.update({'apellido':'Garcia'}) # actualiza la apreja clave-valor
```

Métodos mas utilizados diccionarios

Método	Acción
persona.***keys***()	devuelve las claves del diccionario
persona.***values***()	devuelve los valores del diccionario
persona.***pop***(clave[,default])	elimina la clave del diccionario y este devuelve su valor asociado. Ahora, si no lo encuentra y se indica un valor por defecto, devuelve el valor por defecto indicado
persona.***clear***()	vacía el diccionario
clave in persona	Devuelve True si el diccionario contiene la clave o False en caso contrario
valor in persona.values()	Devuelve True si el diccionario contiene el valor o False en caso contrario

Recorriendo un Diccionario

```
1  persona = {
2           "nombre" : "Juan",
3           "apellido" : "Lopez",
4           "edad": 25,
5           "estatura" : 1.75,
6           "estado_vinculacion-salud" : True
7  }
8  for key in persona:
9    print(key)
10
11 for key in persona:
12   print(persona[key])
13
14 for key, value in persona.items():
15   print("El valor de {} es: {}".format(key, value))
16
17 del persona["edad"]  # elimina el elemento
18
19 # elimina el elemento pero nos devuelve el valor eliminado
20 persona.pop("estatura")
```

Funciones

```
1  def mi_funcion(x):
2    # aquí va el codigo
3    return x
4
5  mi_funcion(x)
```

- Una función es un ***grupo de sentencias*** que se especializan enhacer una ***tarea especifica.***

- Es una buena forma de agrupar codigos y ordenar nuestra aplicacion o programa en ***bloques pequeños.***

- Esto nos facilita la lectura del codigo y la ***reutilización*** del mismo.

Sintaxis

Se escribe la palabra reservada ***def*** seguidamente del nombre de la función y sus parámetros entre paréntesis.

```
1  def saludar(mensaje):
2    # aquí va el codigo
3    print(mensaje)
4
5  saludar("Hola Coders") # llamar a la función
```

Llamando a la función

El llamado de una función, se realiza escribiendo el nombre de la funcion y entre parentesis los parametros (si los hay).

Tipos de parámetros o argumentos

Posicionales: La posición en la que se pasan los parametros si importa.

Con palabra clave: La posición de los parametros no importa, se indica una clave para cada parametro pasado.

```
1  def restar(a,b):
2    # aquí va el codigo
3    resultado = a - b
4    print(resultado)
5
6  restar(10,5) # posición
7  restar(5,10) # posición
8  restar(b=5,a=10) # mediante claves
```

Ámbito de las variables (scope)

El ámbito de una variable(scope) esta ligada a la región del programa donde una variable vive. Fuera del scope de una variable no se podra acceder a ella en lo que respecta a su valor y el manejo de la misma.

Los parámetros y variables que se definan en una función no estaran disponibles o accesibles fuera de la función. **(Ambito Local)**

Las variables que son definidas fuera de la función, son accesibles desde el interior de la función definida. **(Ambito global)**

```
1  def ver_numero():
2    # aquí va el codigo
3    x = 5
4    print(x)
5
6  x = 10
7  ver_numero()  # 5
8  print(x)  # 10
9
```

Clases y Objetos

Python soporta tambien la programación orientada a objetos.

```
1  class Persona:
2    # atributos
3    nombre = "Juan"
4    edad = 25
5
6    # metodos
7    def habla(self):
8      print(self.nombre + " esta hablando")
9
10
11 persona_1 = Persona()
12 persona_1.habla()
13 print(persona_1.edad)
14
```

Una clase es como un modelo, un molde del cual se parte para crear de ella objetos. Este molde (plantilla) contiene la información para definir como serán los objetos, esto es, los atributos y métodos que tendrán.

Desde la clase creada, se pueden crear los objetos que desee. Los objetos de una clase se conocen como **instancias.**

Cada objeto contiene los atributos y métodos de la clase creada, y que se le podra asignar a esos atributos establecidos, unos valores definidos o concretos; *Esto es el estado de un objeto.*

Una funcion dentro de una clase se conoce como método. Las clases contienen un método especial *__init__* el cual es conocido como constructor, dado que sirve para inicializar un objeto. Al momento de crear un objeto, siempre se llama al constructor.

```python
class Persona:
  # atributos
  def __init__(self, nombre, edad):
            self.nombre = nombre
            self.edad = edad
  # metodos
  def habla(self):
    print(self.nombre + " esta hablando")


persona_1 = Persona("Juan", 23)
persona_1.habla()
print(persona_1.edad)

```

Cuando se crea el objeto es necesario indicar los argumentos del constructor.

El parametro ***self*** de los métodos es una referencia a la propia instancia, y es utilizada para acceder a las variables que pertenecen a la clase.

Modulos

Un módulo en Python es un archivo que contiene variables, funciones y clases. De esta forma se puede ordenar y reutilizar el código, y en el que se podra tener acceso al contenido del módulo por los archivos que lo importen.

```
# saludo.py

def saludar():
    print("Hola Coders")
           self.edad = edad

def despedir():
    print("Adios Coders")

```

El acceso a las funciones de otro archivo se da utilizando la palabra reservada ***import***

```
# app.py

import saludo


saludo.saludar()

```

Se puede tambien importar unicamente objetos concretos de un módulo mediante la sintaxis *from ... import:*

```
# app.py

from saludo import despedir


despedir()

```

De esta forma no es necesario escribir el nombre del modulo antes de la utilización de la función.

Se pueden importar varios objetos de un módulo separándolos por una coma.

```
1  # app.py
2
3  from saludo import saludar, despedir
4
5
6  despedir()
7  saludar()
```

Para importar todos los objetos de un módulo, se utiliza un asterisco despues de la palabra reservada *import*

```
1  # app.py
2
3  from saludo import *
4
5
6  despedir()
7  saludar()
```

Lozalización de los modulos.

Al momento de importar un módulo, Python lo buscara de la siguiente forma o bien en los directorios:

1. En el directorio actual (Donde se encuentran los archivos creados)
2. En los directorios declarados en el PYTHONPATH(evariable de entorno que contiene un listado de directorios.
3. En el directorio de instalación de Python por defecto(en Unix Normalmente '/user/local/lib/python')

Hemos llegado al final de este modulo-taller.

Esperemos que sea de su gusto el material plasmado, y los animamos a seguir aprendiendo.

Algunos recursos a consultar para seguir aprendiendo:

https://docs.python.org/3/

https://entrenamiento-python-basico.readthedocs.io/es/latest/

https://www.akademus.es/blog/programacion/principales-usos-python/

https://www.learnpython.org/es/