

Python Básico

Taller

Capítulos 1 al 4

Aprender a utilizar diferentes estructuras de datos es muy importante al momento de realizar cualquier tipo de aplicación.

Python tiene muchos tipos de estructuras de datos, por ejemplos las listas y la tuplas. En algunos lenguajes de programación a estas estructuras se las conocen como arreglos o matrices; y se caracterizan porque los elementos están entre corchetes y separados por una coma. Es importante recordar que los arreglos no son dinámicos el tipo de dato y el tamaño con el cual fue creado no pueden ser modificados.

1. Listas

1.1 Qué son las listas

Una lista es una estructura de datos y un tipo de dato en Python con características especiales. Lo especial de las listas en Python es que nos permiten almacenar cualquier tipo de valor como enteros, cadenas, numéricos e incluso listas.



- Las listas se usan para modelar datos compuestos pero cuya cantidad y valor varían a lo largo del tiempo. Son secuencias mutables y vienen dotadas de una variedad de operaciones muy útiles.
- La notación para una lista es una secuencia de valores encerrados entre corchetes y separados por comas.

```
lista = [1, 2.5, "Python", [5, 6], 4]
```

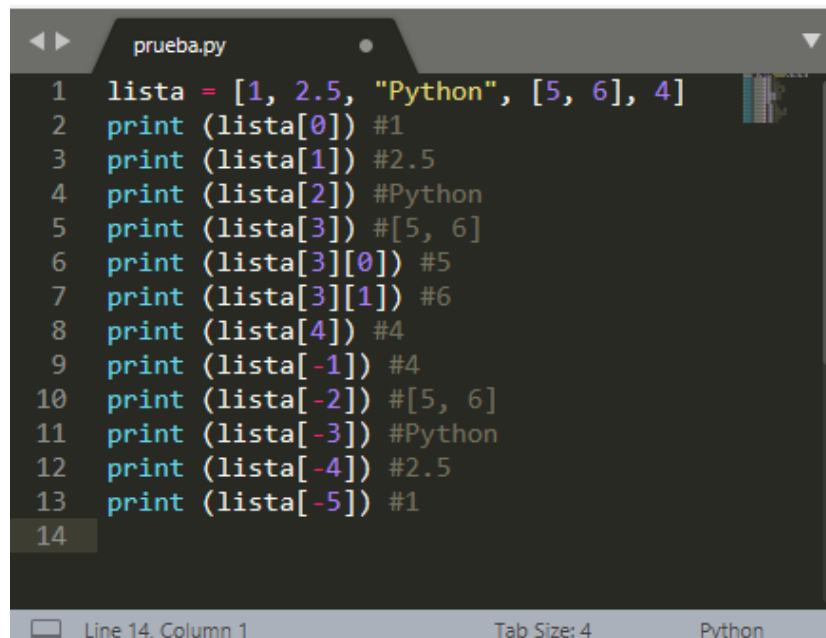
Nota: La diferencia entre arreglos y listas es que estas pueden cambiar de tamaño en tiempo de ejecución (crecer o decrecer)

Recomendación: Aunque se pueden crear listas que mezclen todos los tipos de datos, lo más común es crear listas de solo un tipo.

1.2 Índices

Todos los elementos de la lista pueden ser manejados mediante el índice:

Lista = [[0] [1] [2] [3] [4]]
 1 2.5 "Python" [5, 6] 4

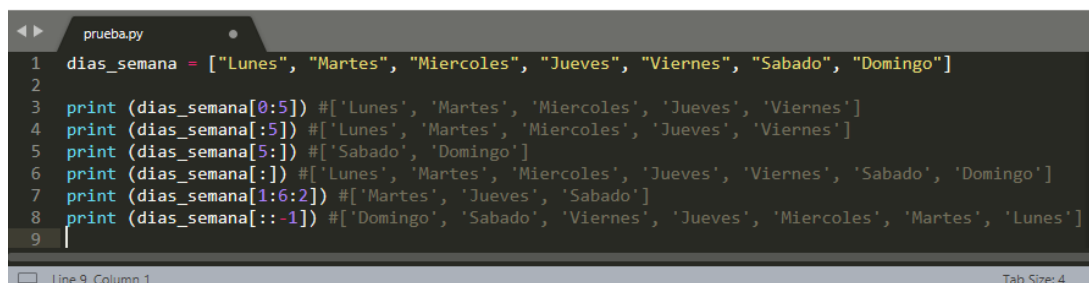


```
prueba.py
1 lista = [1, 2.5, "Python", [5, 6], 4]
2 print (lista[0]) #1
3 print (lista[1]) #2.5
4 print (lista[2]) #Python
5 print (lista[3]) #[5, 6]
6 print (lista[3][0]) #5
7 print (lista[3][1]) #6
8 print (lista[4]) #4
9 print (lista[-1]) #4
10 print (lista[-2]) #[5, 6]
11 print (lista[-3]) #Python
12 print (lista[-4]) #2.5
13 print (lista[-5]) #1
14
```

Nota: Los índices negativos me permiten recorrer la lista comenzando por el último valor.

1.3 Sublistas

Las listas dan mucho de sí; no en vano son uno de las estructuras de datos más importantes de la programación en Python. Una **sublista**, es un trozo de lista de otra mayor.



```
prueba.py
1 dias_semana = ["Lunes", "Martes", "Miercoles", "Jueves", "Viernes", "Sabado", "Domingo"]
2
3 print (dias_semana[0:5]) #['Lunes', 'Martes', 'Miercoles', 'Jueves', 'Viernes']
4 print (dias_semana[:5]) #['Lunes', 'Martes', 'Miercoles', 'Jueves', 'Viernes']
5 print (dias_semana[5:]) #['Sabado', 'Domingo']
6 print (dias_semana[:]) #['Lunes', 'Martes', 'Miercoles', 'Jueves', 'Viernes', 'Sabado', 'Domingo']
7 print (dias_semana[1:6:2]) #['Martes', 'Jueves', 'Sabado']
8 print (dias_semana[::-1]) #['Domingo', 'Sabado', 'Viernes', 'Jueves', 'Miercoles', 'Martes', 'Lunes']
9
```

Estas son las formas en las cuales nosotras podemos crear una nueva lista a partir de otra

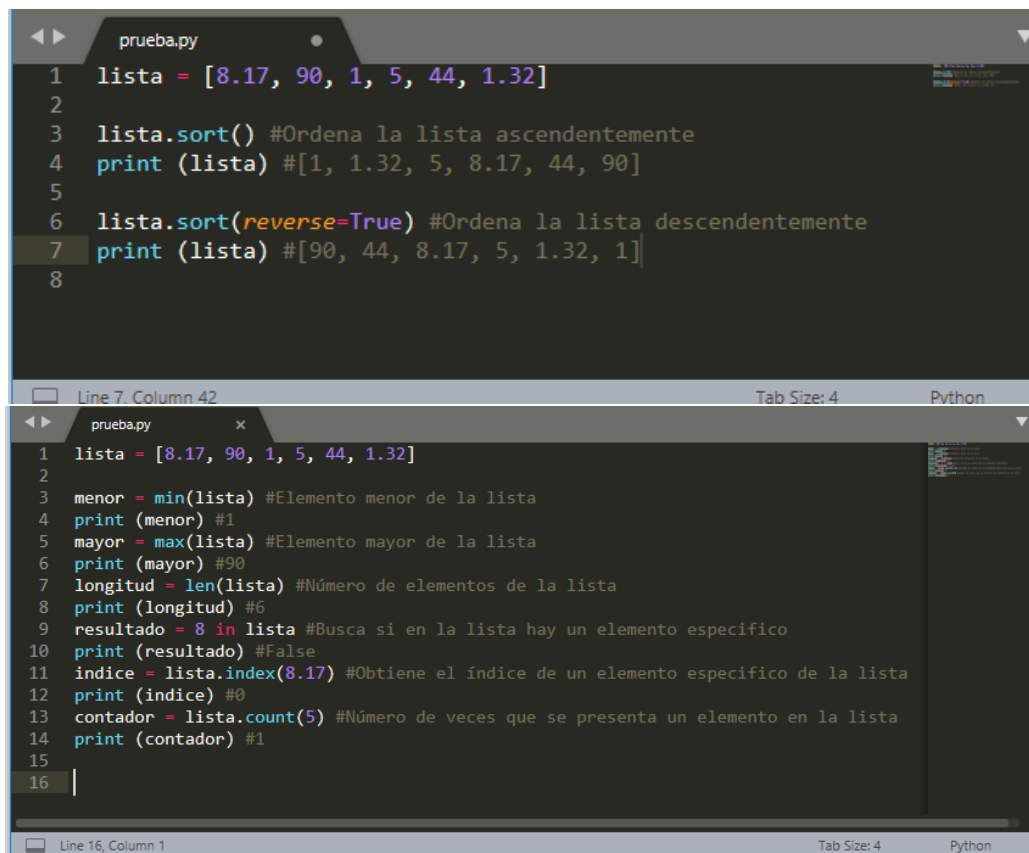
sub = lista[:]	Todos los elementos
sub = lista[start:]	Todos los elementos desde el índice establecido (start)
sub = lista[:end]	Todos los elementos desde el índice cero hasta el índice establecido (end)
sub = lista[start:end]	Todos los elementos de un rango de índices
sub = lista[start:end:step]	Todos los elementos de un rango de índices con saltos

Nota: Este listado es aplicable a las tuplas y los strigs.

1.4 Operadores comunes

lista = [8.17, 90, 1, 5, 44, 1.32]

Si nuestra lista contiene datos de diferente tipo como en el ejemplo anterior es posible querer ordenarla haciendo uso de algoritmos de ordenamiento como quicksort, o bubble sort.



```
prueba.py
1 lista = [8.17, 90, 1, 5, 44, 1.32]
2
3 lista.sort() #Ordena la lista ascendentemente
4 print (lista) #[1, 1.32, 5, 8.17, 44, 90]
5
6 lista.sort(reverse=True) #Ordena la lista descendientemente
7 print (lista) #[90, 44, 8.17, 5, 1.32, 1]
8
```

Line 7, Column 42 Tab Size: 4 Python

```
prueba.py x
1 lista = [8.17, 90, 1, 5, 44, 1.32]
2
3 menor = min(lista) #Elemento menor de la lista
4 print (menor) #1
5 mayor = max(lista) #Elemento mayor de la lista
6 print (mayor) #90
7 longitud = len(lista) #Número de elementos de la lista
8 print (longitud) #6
9 resultado = 8 in lista #Busca si en la lista hay un elemento específico
10 print (resultado) #False
11 indice = lista.index(8.17) #Obtiene el índice de un elemento específico de la lista
12 print (indice) #0
13 contador = lista.count(5) #Número de veces que se presenta un elemento en la lista
14 print (contador) #1
15
16
```

Line 16, Column 1 Tab Size: 4 Python

1.5 Matrices

Una matriz en Python está compuesta por varias listas.

```
prueba.py
1
2 fila_uno = [10, 20]
3 fila_dos = [30, 40]
4
5 matriz = [fila_uno, fila_dos]
6
7
8 print (matriz[0][0]) #10
9 print (matriz[0][1]) #20
10 print (matriz[1][0]) #30
11 print (matriz[1][1]) #40
12
13
Line 2, Column 1 Tab Size: 4 Python
```

Nota: Para matrices de más de una dimensión se podrían anidar las listas.

1.6 Métodos de las listas

Las listas en Python tienen muchos métodos que podemos utilizar, entre todos ellos vamos a nombrar los más importantes. Lista ejemplo:

lista = [2, 5, "Python", 1.2, 5]

- Append()

```
prueba.py
1
2 lista = [2, 5, "Python", 1.2, 5]
3
4 lista.append(10)
5 lista.append("Programar")
6 lista.append([2, 5])
7
8 print (lista) #[2, 5, 'Python', 1.2, 5, 10, 'Programar', [2, 5]]
9
10 |
Line 10, Column 1 Tab Size: 4 Python
```

Se puede agregar cualquier tipo de elemento en una lista, pero tengan en cuenta lo que pasa cuando agregamos una lista dentro de otra, esta lista se agrega como uno y solo un elemento.

- Extend()

Extend también nos permite agregar elementos dentro de una lista, pero a diferencia de append al momento de agregar la lista, cada elemento de esta lista se agrega como un elemento más dentro de la otra lista.

```
prueba.py
1
2 lista = [2, 5, "Python", 1.2, 5]
3
4
5 lista.extend([2, 5])
6
7 print (lista) #[2, 5, 'Python', 1.2, 5, 2, 5]
8
9
10
```

- Remove()

El método remove va a remover un elemento que se le pase como un parámetro de la lista a donde se le esté aplicando.

```
prueba.py
1
2 lista = [2, 5, "Python", 1.2, 5]
3
4 lista.remove(2)
5
6 print (lista) #[5, 'Python', 1.2, 5]
7
8
9
```

Line 8, Column 1

- Reverse()

El método reverse se usa para invertir los elementos de una lista.

```
prueba.py
1
2 lista = [2, 5, "Python", 1.2, 5]
3
4 lista.reverse()
5
6 print (lista) #[5, 1.2, 'Python', 5, 2]
7
8
9
```

Line 8, Column 1 Tab Size: 4 Python

2. Tuplas

2.1 Que son las tuplas

Las tuplas son una estructura de datos muy parecidas a las listas, las cuales también permiten almacenar diferentes tipos de datos.



La diferencia es que las tuplas son inmutables, es decir no se pueden editar los elementos que se encuentren dentro de esta, además que no se puede agregar o quitar elementos (solo es de consulta)

```
tupla = (1, 2.5, "Python", [5, 6], (7, 9))
```

La ventaja de las tuplas sobre las listas es que las tuplas son más rápidas al obtener elementos en cuanto a consulta se refiere.

2.2 Valores por índice

Todos los elementos de la tuplas pueden ser manejados mediante el índice:

	[0]	[1]	[2]	[3]	[4]	
tupla= (1	2.5	"Python"	[5, 6]	(7, 9))

```
prueba.py x
2  tupla = (1, 2.5, "Python", [5, 6], (7, 9))
3  print (tupla[0]) #1
4  print (tupla[1]) #2.5
5  print (tupla[2]) #Python
6  print (tupla[3]) #[5, 6]
7  print (tupla[4]) #(7, 9)
8  print (tupla[-1]) #(7, 9)
9  print (tupla[-2]) #[5, 6]
10 print (tupla[-3]) #Python
11 print (tupla[-4]) #2.5
12 print (tupla[-5]) #1
```

Line 12, Column 21

Generar subtuplas

Una **subtupla** es un trozo de tupla de otra mayor. Estas son las formas en las cuales nosotras podemos crear una nueva tupla a partir de otra.

sub = tupla[:]	Todos los elementos
sub = tupla[start:]	Todos los elementos desde el índice establecido (start)
sub = tupla[:end]	Todos los elementos desde el índice cero hasta el índice establecido (end)
sub = tupla[start:end]	Todos los elementos de un rango de índices
sub = tupla[start:end:step]	Todos los elementos de un rango de índices con saltos

Nota: No es posible cambiar el valor de un elemento de la tupla

tupla[1] = 20 #TypeError

2.3 Comprimir y descomprimir tuplas

```
1  tupla = (1,2,3,4)
2  uno, dos, tres, cuatro = tupla[0], tupla[1], tupla[2], tupla[3]
3  print(uno,dos,tres,cuatro) #1 2 3 4
4  #En Python su puede simplificar la línea anterior:
5  Uno, dos, tres, cuatro = tupla
6  print(uno,dos,tres,cuatro) #1 2 3 4
7
8
```

Nota: Si en la tupla existen seis elementos y solo queremos asignar valores a 4 variables esto es incorrecto.

```
1  tupla = (1,2,3,4,5,6)
2  uno, dos, tres, *cuatro = tupla
3  print(uno,dos,tres,cuatro) #1 2 3 [4, 5, 6]
4  # * guarda todos los elementos restantes en una lista
5  uno, *dos, cinco, seis = tupla
6  print(uno,dos,cinco,seis) #1 [2, 3, 4] 5 6
7
8
```

Generar nuevas tuplas a partir de tuplas y listas

```

1 tupla = (1, 2, 3, 4, 5, 6)
2 lista = [10, 20, 30, 40]
3 resultado = zip(tupla, lista) # función que regresa un objeto tipo zip
4 print(resultado) # <zip object at 0x04DE3DC8>
5 resultado = tuple(resultado) #convertimos a una tupla
6 print(resultado) # ((1, 10), (2, 20), (3, 30), (4, 40))
7 resultado = list(resultado) #obtenemos una lista que contiene tuplas
8 print(resultado) # [(1, 10), (2, 20), (3, 30), (4, 40)]
9
10

```

```

1 tupla = (1, 2, 3, 4, 5, 6)
2 lista = [10, 20, 30, 40]
3 tupla_dos = (100, 200, 300, 400)
4 resultado = zip(tupla, lista, tupla_dos)
5 resultado = list(resultado)
6 print(resultado) # [(1, 10, 100), (2, 20, 200), (3, 30, 300), (4, 40, 400)]
7

```

2.4 Desempaquetado de tuplas

En ciertas ocasiones tendremos la necesidad de obtener *algunos* elementos de nuestras tuplas, por ejemplo, teniendo la siguiente tupla.

```
tupla = (10, 20, 30, 40, 50)
```

Necesito obtener el primero, el segundo y el último elemento; Para lograr esto tendremos un par de opciones; trabajando con índices y sin ellos. Veamos.

Si trabajamos con índices podemos hacerlo lo siguiente.

```

primero = tupla[0]
segundo = tupla[1]
ultimo = tupla[-1]

```


La segunda opción es dejar de trabajar con los índices y utilizar el guión bajo `_`.

```
primero, segundo, _, _, ultimo = tupla
```

Como observamos he colocado dos guiones bajos que hacen referencia a el número `30` y el número `40`, valores que **no** necesitamos, por ende, no necesito almacenarlos en alguna variable; simplemente los ignoramos.

Ahora, que pasa si tengo una tupla mucho más grande y nuevamente necesito obtener esos tres elementos (el primero, el segundo y el último).

```
tupla = (10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400)
```

Lo que podemos hacer es utilizar el guión bajo `_` junto con el asterisco `*` y aplicar lo que hemos visto anteriormente.

```
primero, segundo, *_ , ultimo = tupla
```

De esta forma podemos trabajar de una forma más eficiente con las tuplas.

2.5 De listas a tuplas

```
1 lista = ["taller", "Python", "solo los atrevidos"]
2 tupla = ("introducción", "básico", "listas", "tuplas")
3
4 # tupla → lista
5 nueva_lista = list(tupla) #['introducción', 'básico', 'listas', 'tuplas']
6 print(nueva_lista)
7 # lista → tupla
8 nueva_tupla = tuple(lista) #('taller', 'Python', 'solo los atrevidos')
9 print(nueva_tupla)
10
11 # String → tupla y lista
12 mensaje = "Este es el curso de python"
13 Nueva_lista = list(mensaje)
14 print(Nueva_lista) #['E', 's', 't', 'e', ' ', 'c', 'u', 'r', 's', 'o', ' ', 'd', 'e', ' ', 'p', 'y', 't', 'h', 'o', 'n']
15 # 'c', 'u', 'r', 's', 'o', ' ', 'd', 'e', ' ', 'p', 'y', 't', 'h', 'o', 'n']
16 Nueva_tupla = tuple(mensaje)
17 print(Nueva_tupla) #('E', 's', 't', 'e', ' ', 'c', 'u', 'r', 's', 'o', ' ', 'd', 'e', ' ', 'p', 'y', 't', 'h', 'o', 'n')
18 # 'l', ' ', 'c', 'u', 'r', 's', 'o', ' ', 'd', 'e', ' ', 'p', 'y', 't', 'h', 'o', 'n')
19
20
```

2.6 Importancia de las tuplas

En las tuplas se pueden almacenar datos valiosos como:

- El puerto de un servidor
- La dirección de un folder
- El nombre de un usuario
- Otro dato inmutable

3. CADENAS

3.1 Cadena de caracteres

Manejar texto es importante y muy utilizado. Las cadenas de caracteres son recibidas por los datos que se reciben por teclado, al leer variables de entorno, trabajar con archivos, o al dar formato al texto de salida que el programa provea.

Los string son una cadena de caracteres con un orden ya establecido y también se trabajan a través de índices. Por otro lado es importante saber que los string son inmutables.

<code>sub = string[:]</code>	Todos los elementos
<code>sub = string [start:]</code>	Todos los elementos desde el índice establecido (start)
<code>sub = string [:end]</code>	Todos los elementos desde el índice cero hasta el índice establecido (end)
<code>sub = string[start:end]</code>	Todos los elementos de un rango de índices
<code>sub = string[start:end:step]</code>	Todos los elementos de un rango de índices con saltos

3.2 Trabajo de string como listas

```
1  lenguajes = "py; java; ruby; php; Swift; jscript; c#; c; c++"
2
3
4  Resultado = lenguajes.split() #divide a partir del separador por espacio
5  print(Resultado) #['py;', 'java;', 'ruby;', 'php;', 'Swift;', 'jscript;', 'c#;', 'c;', 'c++']
6
7  Resultado = lenguajes.split(";")
8  print(Resultado) #['py', ' java', ' ruby', ' php', ' Swift', ' jscript', ' c#', ' c', ' c++']
9
10 separador = ";"
11 Resultado = lenguajes.split(separador)
12 print(Resultado) #['py', 'java', 'ruby', 'php', 'Swift', 'jscript', 'c#', 'c', 'c++']
```

3.3 Generar String a partir de una lista

```
1
2 lenguajes = "py; java; ruby; php; Swift; jscript; c#; c; c++"
3
4 Resultado = lenguajes.split() #divide a partir del separador por espacio
5 print(Resultado) #['py;', 'java;', 'ruby;', 'php;', 'Swift;', 'jscript;', 'c#;', 'c;', 'c++']
6
7 Resultado = lenguajes.split(";")
8 print(Resultado) #['py', ' java', ' ruby', ' php', ' Swift', ' jscript', ' c#', ' c', ' c++']
9
10 separador = "; "
11 Resultado = lenguajes.split(separador)
12 print(Resultado) #['py', 'java', 'ruby', 'php', 'Swift', 'jscript', 'c#', 'c', 'c++']
13
14 nuevo_string = " ".join(Resultado)
15 print(nuevo_string) #py java ruby php Swift jscript c# c c++
16
17 #String con saltos de línea
18
19 texto = """ esto es un
20 Texto
21 De
22 Líneas"""
23
24 Resultado = texto.splitlines()
25 print(Resultado) #[' esto es un ', 'Texto ', 'De ', 'Líneas']
```

3.4 Formato para cadena

Darle formato al texto es útil cuando este se mostrará en consola, un pdf o una página web

```
1
2 texto = "taller de python básico"
3
4 resultado = texto.capitalize() #primera mayúscula
5 print(resultado) #Taller de python básico
6
7 resultado = texto.swapcase() #mayúsculas por minúsculas y al revés
8 print(resultado) #TALLER DE PYTHON BÁSICO
9
10 resultado = texto.swapcase() #mayúsculas por minúsculas y al revés
11 print(resultado) #TALLER DE PYTHON BÁSICO
12
13 resultado = texto.upper() #todas mayúsculas
14 print(resultado) #TALLER DE PYTHON BÁSICO
15
16 resultado = texto.lower() #todas minúsculas
17 print(resultado) #taller de python básico
18
19 print(resultado.isupper()) # False
20
21 print(resultado.islower()) # True
22
23 resultado = texto.title() #formato de titulo
24 print(resultado) #Taller De Python Básico
25
26 resultado = texto.replace("python", "ruby") #un string por otro
27 print(resultado) #taller de ruby básico
28
29 resultado = texto.replace("python", "ruby", 1)
30 print(resultado) #taller de ruby básico
31
32 resultado = texto.strip() #sin espacios al inicio o al final
33 print(resultado) #taller de python básico
```

3.5 Formato para cadena parte 2

```
1  curso = "Python"
2
3  nivel = "Básico"
4
5  resultado = "curso de %s %s" %(curso, nivel)
6  print(resultado)
7
8  resultado = "curso de {} {}".format(curso, nivel)
9  print(resultado)
10
11 resultado = "curso de {a} {b}".format(b=nivel, a=curso)
12 print(resultado)
```

```
curso de Python Básico
curso de Python Básico
curso de Python Básico
[Finished in 0.1s]
```

3.6 Concatenación

Qué pasa si queremos modificar uno o más caracteres de nuestro string si son inmutables, podemos generar un nuevo string a través de la concatenación.

```
1  curso = "Curso de Python"
2
3  curso = "c" + curso[1:]
4  print(curso)
5
6  curso = "c" + curso[1:] + " en FIF"
7  print(curso)
8
9  #curso = "c" + curso[1:] + 3 # error ya que solo se puede concatenar str
10
11 curso = "c" + curso[1:] + " " + str(3)
12 print(curso)
```

```
curso de Python
curso de Python en FIF
curso de Python en FIF 3
[Finished in 0.1s]
```

3.7 Búsqueda de cadenas

Saber si un string está dentro de otro

```
1 mensaje = "este es un mensaje en cuanto a longitud de caracteres se refiere"
2
3 resultado = mensaje.count("texto") #cuantas veces texto esta en el string
4 print(resultado)
5
6 resultado = "texto" in mensaje
7 print(resultado)
8
9 resultado = "texto" not in mensaje
10 print(resultado)
11
12 resultado = mensaje.find("texto") #primera posición en la que se encuentra
13 print(resultado)
14
15 resultado = mensaje[resultado: resultado + len("texto")]
16 print(resultado)
17
18 resultado = mensaje.find("codigo") # -1 str no se encuentra
19 print(resultado)
20
21 resultado = mensaje.startswith("Este")
22 print(resultado)
23
24 resultado = mensaje.endswith("e")
25 print(resultado)
```

```
0
False
True
-1

-1
False
True
[Finished in 0.1s]
```

4. DICCIONARIOS

4.1 Que son los diccionarios

Estructura de datos permiten almacenar diferentes tipos de datos incluyendo otros diccionarios. Son mutables podemos modificar sus valores y su tamaño. A diferencia de las listas y las tuplas no podemos extraer sus datos mediante índices. Para ello se requiere de una llave, todos los valores almacenados necesitan tener una llave y cada llave un valor.

Una llave puede ser cualquier objeto inmutable, ejemplo:

```
diccionario = {"total":55} # almacena un 55 con la llave total
```

```
diccionario = {"total":55, "desc":True, "subtotal":15}
```

4.2 Cómo funcionan los diccionarios

```
1  diccionario = {} #diccionario vacio
2  print(diccionario)
3
4  diccionario["nombre"] = "raquel" #agregar llave con su valor
5  print(diccionario)
6
7  valor = diccionario["nombre"] #obtenemos un valor
8  print(valor)
9
10 diccionario["nombre"] = 90
11 print(diccionario)
12
13 dic = {"a":1, "b":2, "c":3, "a":4}
14 print(dic)
```

}

'nombre': 'raquel'}

raquel

'nombre': 90}

'a': 4, 'b': 2, 'c': 3}

Finished in 0.1s]

Nota: No pueden existir llaves duplicadas y si las hubiera la llave tomara el ultimo valor dado

4.3 Obtener elementos de un diccionario

```
1  diccionario = {"a":1, "b":2, "c":3, "a":4}
2  print(diccionario)
3
4  resultado = diccionario["a"]
5  print(resultado)
6
7  #Para saber si una llave existe dentro del diccionario
8
9  resultado = "z" in diccionario
10 print(resultado)
11
12 resultado = diccionario.get("a") # a valor o None
13 print(resultado)
14
15 resultado = diccionario.get("z", "la llave no existe") #segundo parámetro cualquier dato inclusive [] () {}
16 print(resultado)
17
18 resultado = diccionario.setdefault("a", [])
19 print(resultado)
```

{'a': 4, 'b': 2, 'c': 3}

4

False

4

la llave no existe

4

[Finished in 0.1s]

4.4 Llaves, ítems y valores

Es necesario conocer que llaves, que valores hay en un diccionario, para lo cual existen tres métodos:

```
1 #1) Todas las llaves del diccionario
2
3 diccionario = {"a":1, "b":2, "c":3, "d":4, "e":6}
4
5 resultado = diccionario.keys() #objeto dict_keys
6 print(resultado)
7
8 resultado = tuple(resultado)
9 print(resultado)
10
11 #2) Todos los valores del diccionario
12
13 resultado = diccionario.values()
14 print(resultado)
15
16 #3) todas las llaves y valores
17
18 resultado = diccionario.items()
19 print(resultado)
```



```
dict_keys(['a', 'b', 'c', 'd', 'e'])
('a', 'b', 'c', 'd', 'e')
dict_values([1, 2, 3, 4, 6])
dict_items([('a', 1), ('b', 2), ('c', 3), ('d', 4), ('e', 6)])
[Finished in 0.1s]
```

4.5 Eliminar elementos

```

1  #Eliminar llave con su correspondiente valor
2
3  diccionario = {"a":1, "b":2, "c":3, "d":4, "e":6}
4  print(len(diccionario))
5  print(diccionario)
6
7  #Eliminar una llave con su correspondiente valor
8
9  del diccionario["a"]
10 print(diccionario)
11
12 #Para retornar el valor eliminado
13
14 valor = diccionario.pop("b")
15 print(valor)
16 print(diccionario) |
17
18 #Para eliminar todos los elementos del diccionario
19
20 diccionario = {}
21 diccionario.clear()
22 print(diccionario)

```

```

5
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 6}
{'b': 2, 'c': 3, 'd': 4, 'e': 6}
2
{'c': 3, 'd': 4, 'e': 6}
{}
[Finished in 0.1s]

```

5. CICLOS Y CONDICIONALES

5.1 Valores booleanos y valor None

```

1  #Un tipo de dato básico llamado None
2
3  #None para representar la ausencia de algún valor
4
5  variable = None
6  print(variable)
7
8  variable = [1,2,3,4]
9  print(variable)
10
11 #El valor None Python lo toma como un valor false
12 variable = None
13 print(variable and True)
14 print(variable or True)

```

```

None
[1, 2, 3, 4]
None
True
[Finished in 0.1s]

```


5.2 Condicionales

Estructuras de código, en algunas acciones tendremos la necesidad de ejecutar algunos bloques de código dependiendo de ciertos criterios a evaluar, es decir vamos a condicionar nuestro código.

```
1  color_luz = "verde"
2
3  if color_luz == "verde":
4      print("puede continuar")
5  elif color_luz == "amarillo":
6      print("alto parcial")
7  elif color_luz == "rojo":
8      print("alto total")
9  else:
10     print("color no encontrado")
11
12
13  #Las condiciones se realizan de arriba hacia abajo siempre
14  #comenzando en if , el uso del else es optativo:
15
16  color_luz = "rojo"
17
18  if color_luz == "verde":
19      print("puede continuar")
20  elif color_luz == "amarillo":
21      print("alto parcial")
22  elif color_luz == "rojo":
23      print("alto total")
24
25  #considera utilizar valores booleanos
26  variable = True
27
28  if variable:
29      print("el valor es verdadero")
30  else:
31      print("el valor es falso")
```

```
puede continuar
alto total
el valor es verdadero
[Finished in 0.1s]
```

5.3 Ciclo while

```
1 #Podemos ejecutar n cantidad de veces un código hasta que una condición deje de cumplirse
2 #Ejemplo: cuantos dígitos posee un numero
3
4 numero = 123456789
5 contador = 0
6
7 while numero >= 1:
8     contador+=1
9     numero = numero / 10
10 else:
11     print("la cantidad de dígitos es ", contador)
```

```
la cantidad de dígitos es  9
[Finished in 0.1s]
```

5.4 Ciclo for

```
1 #Nos permite iterar un objeto de tal manera que podamos
2 #trabajar con los valores almacenados en este.
3 #Los objetos que nos permiten iterar son las listas,
4 #las tuplas, los strings y los diccionarios
5
6 numeros = [1,2,3,4,5,6,7,8,9,10]
7
8 for numero in numeros: #numero toma el valor de cada elemento en la lista
9     print(numero)
10
11 valores = ( (10, 20), ["text1", "text2"], (True, False))
12
13 for valor in valores:
14     print(valor)
15
16
17 for valor1, valor2 in valores:
18     print(valor1, valor2)
```

```
1
2
3
4
5
6
7
8
9
10
(10, 20)
['text1', 'text2']
(True, False)
10 20
text1 text2
True False
[Finished in 0.1s]
```

5.5 Función Range y enumerate (usados en el ciclo for)

```
1  for valor in range(10):
2      print(valor)
3
4  for valor in range(-1, 20):
5      print(valor)
6
7  for valor in range(1, 101, 2):
8      print(valor)
9
10 lista = [1,2,3,4,5,6]
11
12 for indice in range(len(lista)):
13     print("indice", indice, "valor: ", lista[indice])
14
15 #Con la función enumerate podemos recorrer un objeto iterable,
16 #en cada ciclo de iteración podemos hacer uso de dos valores
17
18 lista = [1,2,3,4,5,6]
19
20 for indice, valor in enumerate(lista, 10):
21     print("indice", indice, "valor: ", valor)
22
23 #índice es una variable que siempre se inicia en 0 y aumenta en 1 en cada iteración
24 #la segunda variable va a contener los valores del objeto iterable
```

5.6 Break y continue

```
1  #Permiten modificar el comportamiento de los ciclos
2
3  titulo = "Curso de Python Basico"
4
5  for caracter in titulo:
6      print(caracter)
7
8
9  for caracter in titulo:
10     if caracter == "P":
11         break
12     print(caracter)
13 #break -> Salimos del ciclo
14
15 for caracter in titulo:
16     if caracter == "P":
17         continue
18     print(caracter)
19 #continue -> Saltamos a la siguiente iteración
```

5.7 Asignación de valores mediante if

```
Prueba.py x
1  #Asignar valo a una variable meddinte condiciones
2  #Colocar sobre una pagina web la calificación de un alumno,
3  #la calificación estará de color verde si y solo si cuenta con
4  #una calificación aprobatoria >=7 si la calificación es desaprobatoria
5  #debe estar en rojo
6
7  calificacion = 7
8  color = None
9
10 if calificacion >= 7:
11     color = "verde"
12 else:
13     color = "rojo"
14
15 print(calificacion,color)
16
17 calificación = 5
18 color = None
19 color = "verde" if calificacion >=7 else "rojo"
20 print(calificación, color)

7 verde
5 verde
[Finished in 0.1s]
```