Python - Introducción

Computación – Curso 2024

Prof. Jorge Runco

Básico

- Se pueden crear asignando un valor a un nombre sin necesidad de declararla antes.
- X = 4
- X = 2.15
- Para la asignación se usa el signo = . Para una comparación se usa ==
- Operaciones: + - * / %
- + también se usa en concatenación de strings
- % para formato (como en fprintf)
- and, or, not son operadores lógicos. No se usan símbolos.

- Comentarios:
- El símbolo # al inicio de cada una de las líneas que se desean comentar.
- # Esto es un comentario en Python
- También usar tres comillas dobles(""") al inicio y al final de cada línea que se desea comentar.
- “””Esto es un bloque de código comentado en Python”””

Estructuras de control

if

- if (condición) :
  sentencia 1
  sentencia 2
  sentencia 3
- La sentencia if debe ir terminada por : y el bloque de código a ejecutar debe estar identado.
- Así sentencia 1 y sentencia 2, están dentro del if.
- Y no sentencia 3

else y elif

```
if  condición :
    print(“Se cumple condición")
else:
    print("No se cumple condición")
```

- Igual que antes, si se cumple la condición se ejecuta el bloque del if.
- Si no se cumple la condición, se ejecuta el bloque del else.

else y elif

```
a = 3
if a == 1:
    print("Es 1")
elif a == 2:
    print("Es 2")
elif a == 3:
    print("Es 3")
```

Vimos como ejecutar un bloque de código si se cumple una condición, u otro si no se cumple la condición.
Podemos tener varias condiciones diferentes y para cada una queremos un código distinto.
Entonces usamos elif como muestra el ejemplo.
Se pueden usar tantos elif como se desee.

Elif y else

```
a = 5
if a == 5:
    print("Es 5")
elif a == 6:
    print("Es 6")
elif a == 7:
    print("Es 7")
else:
    print("Es otro")
```

Se pueden usar el if con el elif y un else al final. Notar que if y else solamente puede haber uno, mientras que elif puede haber varios.
Otros lenguajes de programación tienen la sentencia case en lugar de elif.
Python no tiene esta sentencia.

for

- Igual que en otros lenguajes, se conoce el número de veces que se ejecuta
- Ej. for i in range(0, 5):

```
        print(i)     #0, 1, 2, 3, 4
```

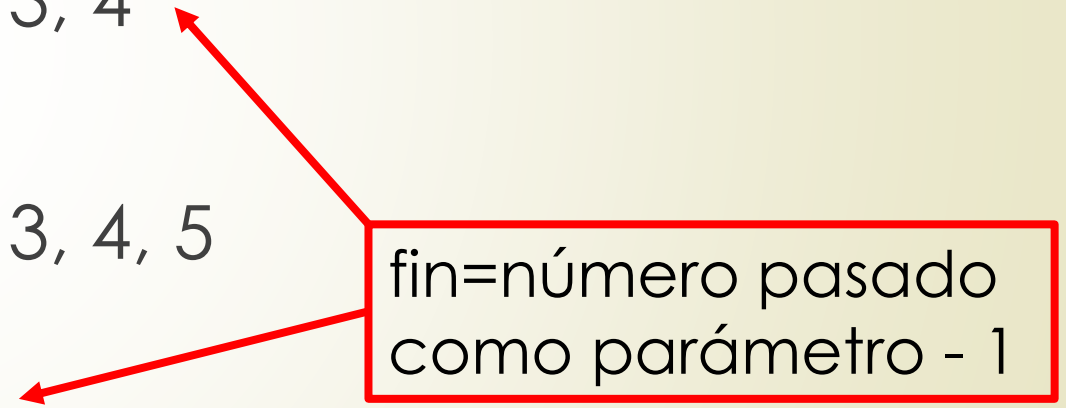
- Ej. for i in (0, 1, 2, 3, 4, 5):

```
        print(i)     #0, 1, 2, 3, 4, 5
```

- Ej. for i in range(6):

```
        print(i)     #0, 1, 2, 3, 4, 5
```

- range(inicio, fin, salto)



- Ej. for i in range(5, 20, 2):

  print(i) #5,7,9,11,13,15,17,19

- Ej. for i in range (5, 0, -1):

  print(i) #5,4,3,2,1

- for <variable> in <iterable>:

  <Código>

- Los iterables son objetos que pueden ser iterados o accedidos con un índice. Como las listas, tuplas, cadenas o diccionarios.

- Ej. for i in "Python":

- print(i)

- P y t h o n

break

- for num in range(10):

```
        if num == 5:
            print ("Se cumple la condición que finaliza el bucle ")
            break
        print(f" El número actual es {num}")
print("Continuamos después del bucle")
```

- Respuesta:

El número actual es 0

El número actual es 1

El número actual es 2

El número actual es 3

El número actual es 4

Se cumple la condición que finaliza el bucle

Continuamos después del bucle

continue


```
for num in range(10):
    if num == 5:
    continue
    print(f"El número actual es {num}")
print("Continuamos con el bucle")
```

- Respuesta

```
El número actual es 0
El número actual es 1
El número actual es 2
El número actual es 3
El número actual es 4
El número actual es 6
El número actual es 7
El número actual es 8
El número actual es 9
Continuamos con el bucle
```

Funcionamiento similar al break. Se inicia el bucle y se revisa si la condición es true o false. Normalmente, el bucle itera hasta que la condición se convierte en false. Sin embargo, en el caso continue, se hace una pregunta "en la mitad". Si la respuesta no cumple la condición adicional, el bucle continúa con normalidad. Sin embargo, si la condición de continue se cumple, el bucle vuelve al principio y se ejecuta con un nuevo valor.

while

- El while se ejecuta mientras sea verdadera la condición.

```
a = input("Ingresar un número: ")
aentero= int(a)
x = 0
while x < 5:
    print(aentero)
    x= x +1
```

inpunt y print

- La función input() tiene el propósito de recibir un valor desde el teclado.
- ```
  a = input("Ingresar un número: ")
  aentero= int(a)
  ```
- La función input "toma" como una cadena de caracteres el valor ingresado.
- Si queríamos ingresar un número, debemos convertirlo al tipo de dato deseado.
- La función print() imprime en pantalla. El parámetro de esta función puede ser de cualquier tipo, automáticamente es convertido a string.
  ```

print

- La función print() imprime en pantalla. El parámetro de esta función puede ser de cualquier tipo, automáticamente es convertido a string.
- >>>print('texto')

  texto
- >>>cadena = "Esto es una cadena"

  >>>print(cadena)

  Esto es una cadena
- >>>print(1 + 2)

  3
- >>>a = 4 * 5

  print(a)

  20

print – Cadenas f

- Hace más sencillo introducir variables y expresiones en las cadenas. Una cadena "f" contiene variables y expresiones entre llaves "{}" que se sustituyen directamente por su valor. Las cadenas "f" se reconocen porque comienzan por una letra f antes de las comillas de apertura.
- nombre = “Jorge"
- edad = 65
- print(f"Me llamo {nombre} y tengo {edad} años.")

Funciones

- def mio():

  print("Hola!")

- La palabra clave def
- Un nombre de función
- Paréntesis (), y dentro de los paréntesis los parámetros de entrada, aunque los parámetros de entrada pueden ser opcionales.
- Los dos puntos :
- Algún bloque de código para ejecutar
- Una sentencia de retorno (opcional)

- # función con un parámetro

```
def mia(nombre):
        print("Hola " + nombre + "!")
mia("Jorge")  # llamada a la función, 'Hola Jorge!' se muestra en la pantalla
```

- # función con múltiples parámetros con una sentencia de retorno

```
def multiplica(val1, val2):
        return val1 * val2
multiplica(3, 5)  # muestra 15 en la pantalla
```

```
def suma(a, b):
    return a + b


result = suma(1, 2)
# result = 3
```

Archivos

Lectura y escritura

Lectura

- Podemos abrir un archivo con la función open() pasando como argumento el nombre del archivo que queremos abrir.
- fid = open('ejemplo.txt')
- Podemos leer su contenido con read()
- print(fid.read())

  Contenido de la primera línea

  Contenido de la segunda línea

  Contenido de la tercera línea

  Contenido de la cuarta línea

- Se puede leer un número de líneas determinado y no todo el archivo junto. Se usa la función readline()
- fid = open('ejemplo.txt')

```
print(fid.readline())
print(fid.readline())
```

Contenido de la primera línea

Contenido de la segunda línea

- Si le pasamos un número como argumento, se leerá un determinado número de caracteres. En el siguiente ejemplo, se lee todo el archivo carácter por carácter.

```
fid = open('ejemplo.txt')
caracter = fid.readline(1)
while caracter != "":
    print(caracter)
    caracter = fid.readline(1)
```

Argumentos de open()

- ‘r’: Por defecto, para leer el fichero.
- ‘w’: Para escribir en el fichero.
- ‘x’: Para la creación, fallando si ya existe.
- ‘a’: Para añadir contenido a un fichero existente.
- ‘b’: Para abrir en modo binario.
- fid = open('ejemplo.txt', 'r')

Escritura

- 'w': Borra el archivo si ya existiese y crea uno nuevo con el nombre indicado.
- 'a': Añadirá el contenido al final del archivo si ya existiese (append).
- 'x': Si ya existe el archivo se devuelve un error.
- Ej. fid = open('guardar_datos.txt', 'w')

  fid.write('Contenido a escribir')

  fid.close()

Tipo de datos

- Numéricos
- De texto
- Booleanos
- De secuencias
- De mapeo
- Conjuntos
- De bytes

Datos numéricos

- Enteros **int**
- Punto ó coma flotante **float.** Números reales.
- Complejos **complex.**

Datos tipo texto

- **string** cadena ó secuencia de caracteres. Los datos tipo string se representan entre comillas simples ('') o dobles ("").
- Las cadenas de caracteres son inmutables. Es decir, no se pueden modificar una vez creadas. Si hay que cambiarlas, se hace una nueva con los cambios deseados.

Datos booleanos

- Booleanos **bool**. True ó False. Se utilizan principalmente en expresiones lógicas y de control de flujo.

Tipos de secuencias

- Tipo **list** (listas).Las listas son una estructura de datos que permite almacenar múltiples valores y acceder a ellos por su posición en la lista.
- Los datos tipo lista van entre corchetes [ ], que contienen los elementos de la lista separados por comas. Los elementos pueden ser de cualquier tipo de datos.

- Por ejemplo:
- [1, 2, 3, 4, 5], ['a', 'b', 'c'], [1, 'a', True, [2, 3, 4]]
- Las listas en Python son mutables, significa que se pueden modificar una vez que se han creado. Esto permite agregar, eliminar o modificar elementos en una lista.
- Datos tipo **tuple** (tuplas). Son similares a las listas, pero a diferencia de las listas, son inmutables. Las tuplas son una estructura de datos que permite almacenar múltiples valores y acceder a ellos por su posición en la tupla. Pero una vez creada la tupla, los elementos no pueden modificarse.

- Los datos tipo tupla se representan mediante paréntesis ( ) que contienen los elementos de la tupla separados por comas. Los elementos pueden ser de cualquier tipo de datos.
- "Por ejemplo:
- (1, 2, 3, 4, 5), ('a', 'b', 'c'), (1, 'a', True, (2, 3, 4))
- Las tuplas en Python son inmutables; es decir, que no se pueden modificar una vez que se han creado.
- Si es necesario cambiar los datos, usamos una una lista en lugar de una tupla.

- Datos tipo **range.** Representan una secuencia inmutable de números enteros.
- Se utiliza comúnmente para generar una secuencia de números enteros para su uso en un bucle for.
- La función range() devuelve un objeto de tipo range. El objeto range toma tres argumentos: el valor inicial, el valor final (no incluído en la secuencia) y el tamaño del paso. Por defecto, el valor inicial es 0 y el tamaño del paso es 1.
- Por ejemplo:
- range(0, 10, 1), range(0, 10)

- El primer ejemplo crea un objeto range que representa la secuencia de números enteros del 0 al 9 (inclusive) con un tamaño de paso de 1. El segundo ejemplo es equivalente al primer ejemplo, ya que el valor inicial y el tamaño del paso se asumen como 0 y 1, respectivamente, por defecto.

Tipo de datos de mapeo

- Datos tipo **dict** (diccionario), son una estructura de datos que permite almacenar un conjunto de datos como pares clave-valor, donde cada valor es accesible a través de una clave única. En otras palabras, los diccionarios permiten asociar valores con claves.

- Ejemplo
- d = {'nombre': ‘Jorge', 'edad’: 65}
- print(d['nombre’])
- Jorge
- Es posible modificar los valores en un diccionario, agregar nuevos pares clave-valor y eliminar pares clave-valor existentes.
- A diferencia de los tipos secuenciales (list, tuple o range), que son indexados por un índice numérico, los diccionarios son indexados por claves.

- Podemos pensar en un diccionario como un contenedor de pares clave: valor, en el que la clave puede ser de "cualquier tipo" y es única en el diccionario que la contiene.
- La forma más simple, es encerrar una secuencia de pares clave: valor, separadas por comas entre llaves {}
- d = {1: 'hola', 89: ‘Vamos', 'a': 'b', 'c': 27}
- En el diccionario anterior, los enteros 1 y 89 y las cadenas 'a' y 'c' son las claves.
- Para acceder al dato
- d[‘c’]
- 27

- >>> d = {'uno': 1, 'dos': 2, 'tres': 3}
- >>> d['dos']
- 2
- También existe el método get(clave[, valor por defecto]). Devuelve el valor correspondiente a la clave clave. Si la clave no existe no marca ningún error, sino que devuelve el segundo argumento valor por defecto. Si no se proporciona este argumento, se devuelve el valor None.

- >>> d = {'uno': 1, 'dos': 2, 'tres': 3}
- >>> d.get('uno')
- 1
- # Devuelve 4 como valor por defecto si no encuentra la clave
- >>> d.get('cuatro', 4)
- 4

Tipo de datos de conjuntos

- Datos tipo **set**, son una colección de elementos únicos e inmutables; es decir, no debe haber duplicados en un conjunto y no se pueden cambiar después de ser creados. Un conjunto se crea utilizando llaves { } o la función set() y los elementos se separan por comas.