CQ6

Definición y ejemplo

False: Operador lógico de tipo boleano, es decir que puede tomar solo dos valores: “True” y “False”, puede ser usado para comparar, negar o afirmar.

True: Operador lógico de tipo boleano, es decir que puede tomar solo dos valores: “True” y “False” , puede ser usado para comparar, negar o afirmar.

```
3 > 2
True
3 < 2
False
```

None: Es usado cuando se requiere crear una variable, esto, porque Python no diferencia entre crear o asignar variables.

```
a = None
if a == None:
   print("a es None.")
else:
   print("a no es None.")
```

And: Operador lógico, con el cual se puede comprobar una comparación entre dos variables, también verificar el resultado de las funciones establecidas y realizar una determinada acción dependiendo del resultado.

```
func1 = True
func2 = False

if func1 and func2:
   print('Both function executed successfully')
else:
   print("Task failed")
```

As: Un patrón AS coincide con un patrón OR a la izquierda de la palabra clave “as” con un sujeto. En pocas palabras llega a convertir en un condicional.

```
as_pattern ::=  or_pattern "as" capture_pattern
```

Assert: Instrucción de python que me permite definir condiciones que deban cumplirse siempre.

```
a = otra_funcion()
assert a >= 0       # Asegurarse de que es positivo
return a + 1
```

Break: Le proporciona la oportunidad de cerrar un bucle cuando se activa una condición externa.

```
number = 0

for number in range(10):
    if number == 5:
        break    # break here

    print('Number is ' + str(number))

print('Out of loop')
```

Class: Se usa para crear una clase y podeos añadirle tributos a esta.

```
class MiClase:
    atributo1 = "valor1"
    atributo2 = "valor2"
```

Continue: Omitir la parte de un bucle en la que se activa una condición externa, pero continuar para completar el resto del bucle.

```
number = 0

for number in range(10):
    if number == 5:
        continue    # continue here

    print('Number is ' + str(number))

print('Out of loop')
```

Def: Definición de función usada para crear objetos funciones definidas por el usuario.

```
>>> def resta(a, b):
...     return a - b
>>> resta(30, 10)
20
```

Elif: Se pueden verificar varias condiciones al incluir una o más verificaciones

```
x=int(input(" Ingrese primer número "))
y=int(input(" Ingrese segundo número "))
if int(y==0):
  print("Error")
elif y>0:
  print(x/y)
  if int(x%y)==0:
    print("la división es exacta")
  else:
    print("La división no es exacta")
    print("El residuo de la división es ",x%y)
```

Else: Es una posible opción en caso de que no se cumpla alguna o todas las anteriores.

```
  if int(x%y)==0:
    print("la división es exacta")
  else:
    print("La división no es exacta")
    print("El residuo de la división es ",x%y)
```

Except: Se usa para lanzar una excepción sin que el programa sea detenido.

```
a = 5; b = 0
try:
    c = a/b
except ZeroDivisionError:
    print("No se ha podido realizar la división")
```

Finally: Se ejecuta siempre, haya o no haya habido excepción.

```
try:
    x = 2/0
except:
    print("Entra en except, ha ocurrido una excepción")
finally:
    print("Entra en finally, se ejecuta el bloque finally")
```

For: Se usa para iterar sobre los elementos de una secuencia (como una cadena de caracteres, tupla o lista) u otro objeto iterable:

```
nums = [4, 78, 9, 84]
for n in nums:
print(n)
4
```

78
9
84

From: Se usa para especificar de dónde se desea adquirir algo, en el caso de combinarlo con import, para especificar de dónde obtendré esa referencia.

```
>>> globals()['os']
<module 'os' from '/System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/os.pyc'>
>>> locals()['os']
<module 'os' from '/System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/os.pyc'>
>>> globals()['os.path']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'os.path'
>>>
```

Global:

If: Se usa para la ejecución condicional:

```
e=int(input("Ingrese un número "))
r=int(input("Ingrese otro número "))
if int(e%r==0):
  print("Son múltiplos")
else:
  print("No son múltiplos")
```

Import: Crea una referencia a ese módulo en el espacio de nombres actual**.**

```
>>> import sys
>>> import os.path
```

In: Devuelve True si un elemento se encuentra dentro de otro.

```
a = [3, 4] 3 in a
```

```
True
```

Is: Devuelve True si los elementos son exactamente iguales.

```
x = 10
y = 10
x is y
```

```
True
```

Lambda: Las funciones Lambda se comportan como funciones normales declaradas con la palabra clave def. Resultan útiles cuando se desea definir una función pequeña de forma concisa.

```
square = lambda x: x ** 2
print(square(3))  # Resultado: 9

def square1(num):
   return num ** 2
print(square(5))  # Resultado: 25
```

Nonlocal: Indica que la variable dada sobre la que se trabaja no pertenece localmente a la función anidada dada.

```
a = 0

def outr():

    a = 1

    def innr():

        nonlocal a

        a = 2

        print("inner variable value:", a)

    innr()

    print("outer variable value:", a)

outr()

print("global variable value:", a)
```

Not: El operador “not” es unario, de negación. Devuelve el valor opuesto la valor booleano.

```
>>> not True
```

```
False
```

Or: Se utliza para agregar una condición a otra, es decir que si se cumple lo primero o lo segundo se realiza la siguiente acción.

```
if Carlos_edad < Hombres_edad or Juan_edad < Hombres_edad or Marita_edad < Mujeres_edad:
print ("Alguno no cumple la condicion de edad, no pueden entrar juntos")
```

Pass: Le indica al programa que ignore esa condición y continúe ejecutando el programa como de costumbre.

```
number = 0

for number in range(10):
    if number == 5:
        pass    # pass here

    print('Number is ' + str(number))

print('Out of loop')
```

Raise: Se utiliza para lanzar una excepción o levantar una excepción.

```
raise NameError("Información de la excepción")
```

Return: Indica el final de la función pero también el valor que devuelve la función.

```
def escribe_media():
    media = (a + b) / 2
    print(f"La media de {a} y {b} es: {media}")
    return

a = 3
b = 5
escribe_media()
print("Programa terminado")
```

Try: Es especifica para gestionar excepciones o código de limpieza para un grupo de sentencias:

```
try:
    res = dividir(num, div)
    print(res)
except ZeroDivisionError:
    print("Trataste de dividir entre cero :( ")
```

While: Se usa para la ejecución repetida siempre que una expresión sea verdadera:

```
while numero <= 10:

    suma = numero + suma

    numero = numero + 1

print ("La suma es " + str(suma))
```

With: Se usa para ajustar la ejecución de un bloque con métodos definidos por un administrador de contexto (ver sección Gestores de Contexto en la Declaración with).

```
with A() as a, B() as b:
    SUITE
```

Yield: Se le llama así a un objeto que es capaz de retornar sus miembros uno a la vez. Entre los iterables se encuentran estructuras de secuencia como son:

```
def square(nums):
     for i in nums:
          yield(i*i)

someNums = square([1, 2, 3, 4, 5])

print someNums
```