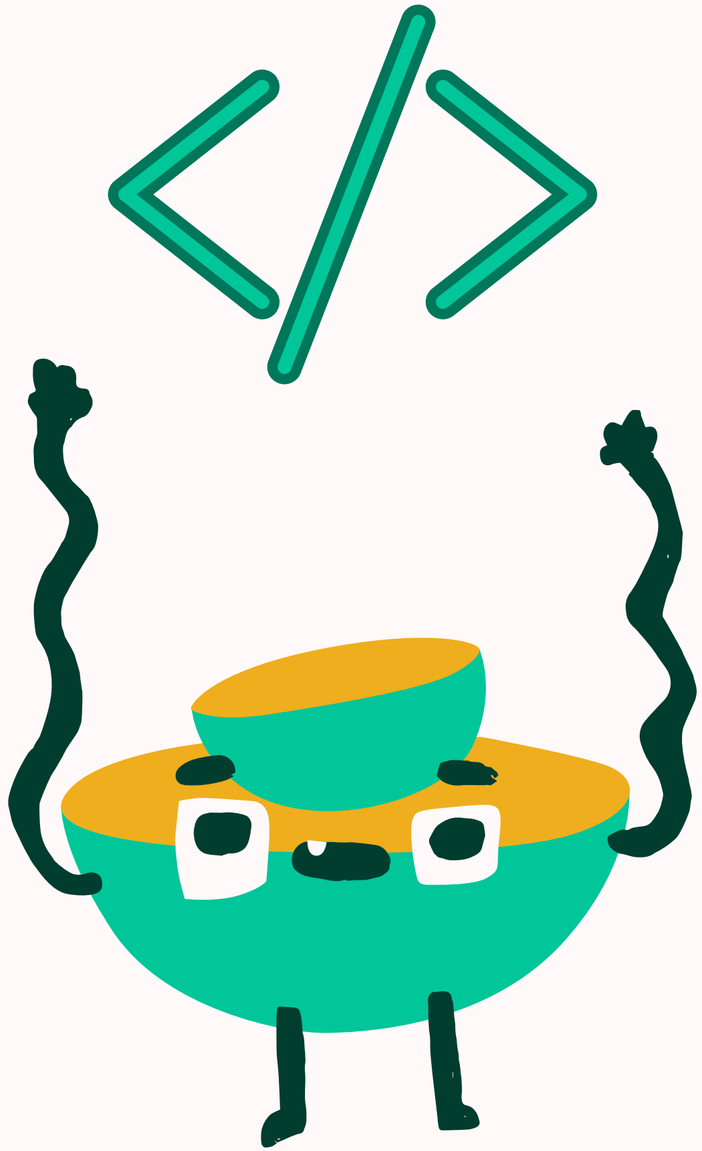
ERRORES TÍPICOS AL PROGRAMAR

CC-1002



RETURN ≠ PRINT

Return

El return nos indica el fin de una función y le sigue el valor que queremos obtener a partir de la función

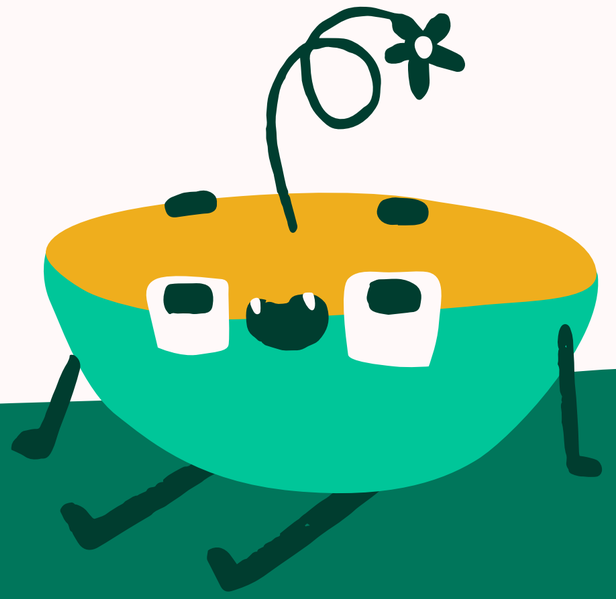
Print

Solo imprime en la ventana lo que le indiquemos entre los parentesis

```python
def saludo():
  return 'Hola Mundo'

def saludos():
  print('Hola Mundo')

a = saludo()
b = saludos()
```

¿Cuál es la diferencia entre a y b?

hint: el tipo



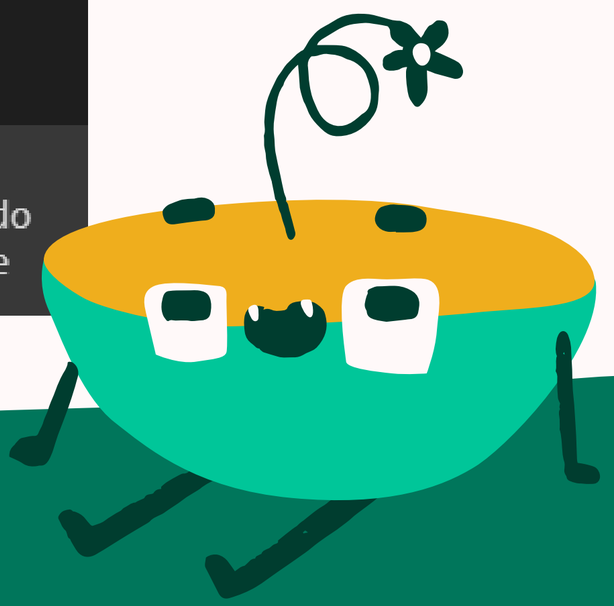
RETURN ≠ PRINT

La variable a contiene el str 'Hola mundo' mientras que la variable b contiene un None

```
def saludo():
  return 'Hola Mundo'

def saludos():
  print('Hola Mundo')

a = saludo()
b = saludos()

print('La variable es del tipo', type(a),'La variable contiene:',a)
print('La variable es del tipo', type(b),'La variable contiene:',b)
```

```
Hola Mundo
La variable es del tipo <class 'str'> La variable contiene: Hola Mundo
La variable es del tipo <class 'NoneType'> La variable contiene: None
```

La variable a es del tipo str, mientras que la b es un NoneType



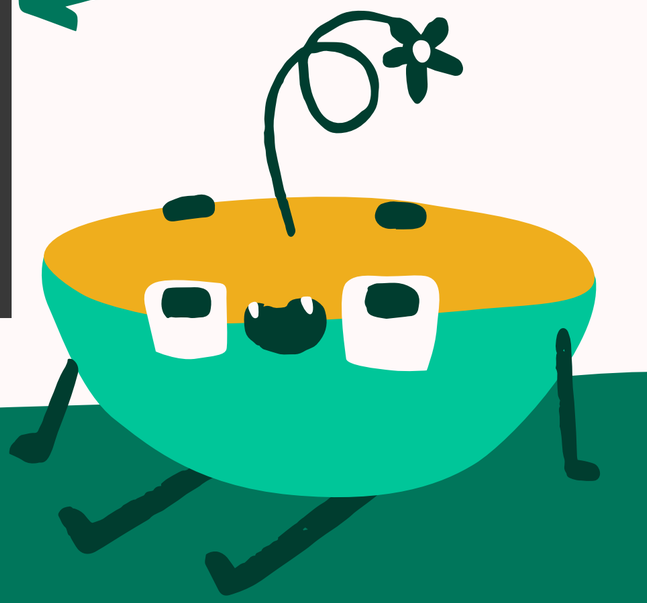
RETURN ≠ PRINT

```python
def saludo():
  return 'Hola Mundo'

def saludos():
  print('Hola Mundo')

a = saludo()
b = saludos()

print(a+' ¿Cómo están?')
print(b+'¿Cómo están?')
```

```
Hola Mundo
Hola Mundo ¿Cómo están?
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-e6a18a8a459b> in <module>()
      9
     10 print(a+' ¿Cómo están?')
---> 11 print(b+'¿Cómo están?')

TypeError: unsupported operand type(s) for +: 'NoneType' and 'str'
```

Si quisiéramos usar b para hacer un print(b + '¿Cómo están?') nos arrojaría error, ya que estamos concatenando un None con un str

Por otro lado, al hacer el mismo print con la variable a, obtenemos el resultado que queríamos



PRINT

Separado por coma

Cuando escribimos un print separado por comas no importa el tipo de variable que le estemos ingresando y todas serán impresas con espacios de separación

Separado por +

Cuando escribimos el print mezclando cosas con un '+' todos los elementos deben ser del tipo str y se deben escribir los espacios de separación

```
a = 2
print('Hola mundo ' + str(a))
print('Hola mundo' + str(a))
print('Hola mundo',a)
```

```
Hola mundo 2
Hola mundo2
Hola mundo 2
```

En el primer print se explicitan los espacios mientras que en el segundo no

IMPORTAR MODULOS

from modulo import funcion

Al escribir este comando se va a importar solo la función indicada desde el módulo con el nombre original

```
from random import randint

a = randint(1,4)
print(a)
```

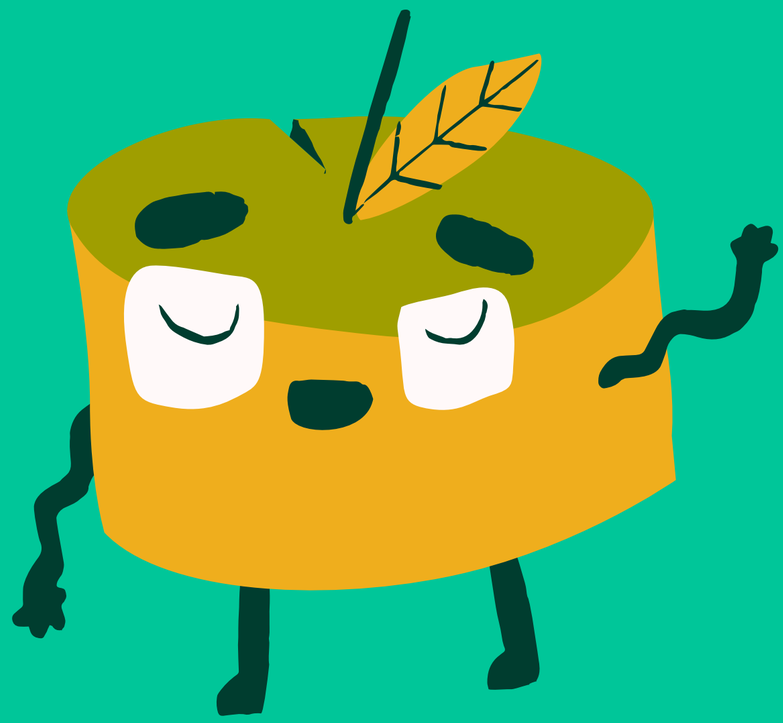
```
4
```

from modulo import *

Con este comando se van a importar todas las funciones desde el módulo indicado y se pueden usar con los mismos nombres que estan en el modulo

```
from random import *

a = randint(1,4)
print(a)
```

```
3
```



IMPORTAR MODULOS

import modulo

Con este comando se importará el módulo completo y para usar sus funciones se deberá referenciar el módulo antes de usarla

```
import random

a = random.randint(1,10)
print(a)
```

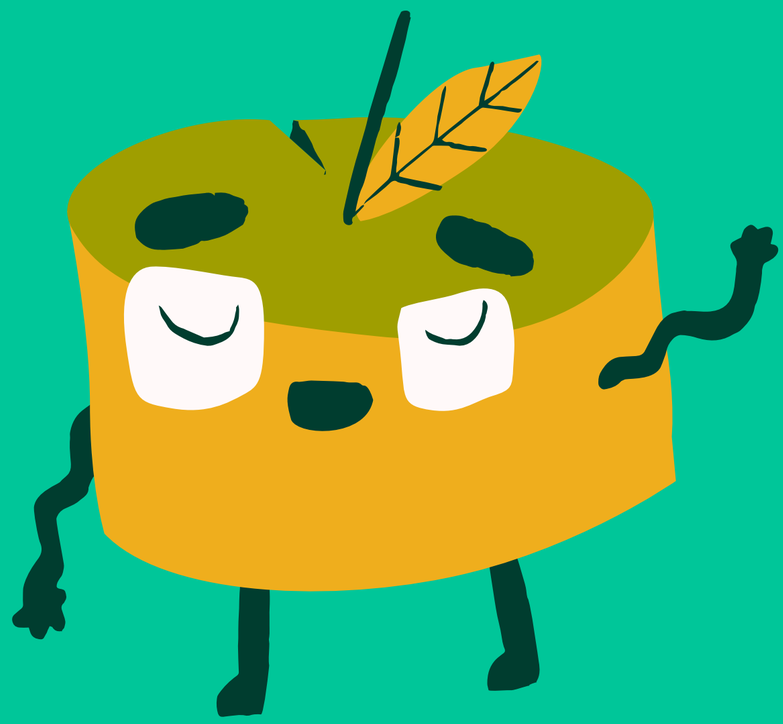
```
2
```

import modulo as mod

Al importar un módulo así, pasa lo mismo que en el caso anterior con la pequeña diferencia que tendremos que escribir el nombre que le asignamos para referenciarlo

```
import random as rn

a = rn.randint(1,10)
print(a)
```

```
7
```



TIPO DE UN INPUT

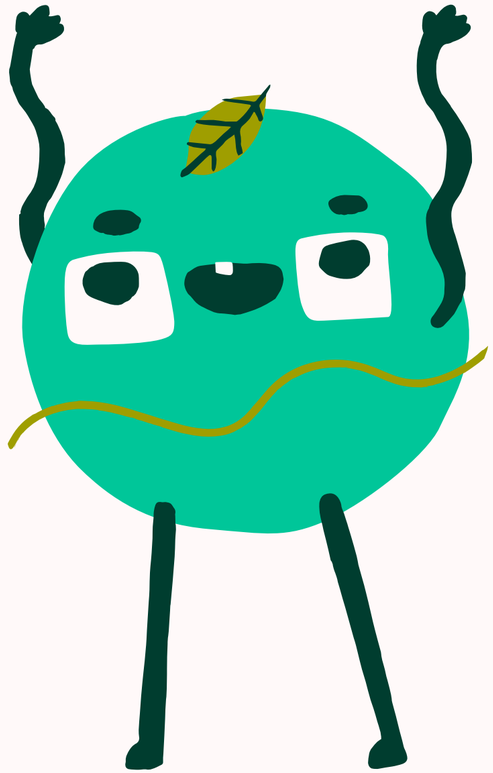
```
a = input('ingrese numero: ')
b = int(input('ingrese numero: '))
```

El tipo de la variable a va a ser un str aunque se le ingrese un número, mientras que el tipo de la variable b si va a ser un número

Ingreso del input

Cada vez que usen un input para interactuar con el usuario, lo que este escriba se va a recibir como un str.

En el caso de que usted necesite un int y lo este obteniendo a través de un input, es necesario que le cambie el tipo de variable que es

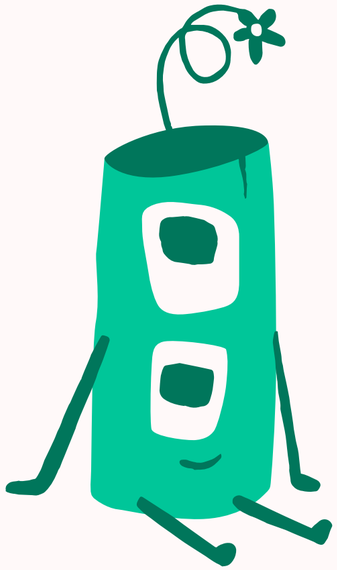


Condiciones

La pregunta aquí correspondería a si el número es menor a algunos de los tres valores indicados y en caso de serlo, cuál es el menor número que es mayor que n entre el 10, el 20 y el 30

```
def menor(n):
  if n<10:
    print('menor a 10')
  elif n<20:
    print('menor a 20')
  elif n<30:
    print('menor a 30')

menor(7)
```

```
menor a 10
```



Si bien 7 también es menor a 20 y 30, solo imprime que es menor a 10 por la forma en la que escribimos el código

if

Se usa para evaluar una condición que queremos que sea verdadera. Corresponde a una de las posibles respuestas de una pregunta

elif

Continúa con la 'pregunta' que se hizo en el if y da una posible segunda, tercera, cuarta, etc. respuesta a la pregunta indicada

else

Corresponde a todos los casos que no se mencionaron en los if o elif anteriores

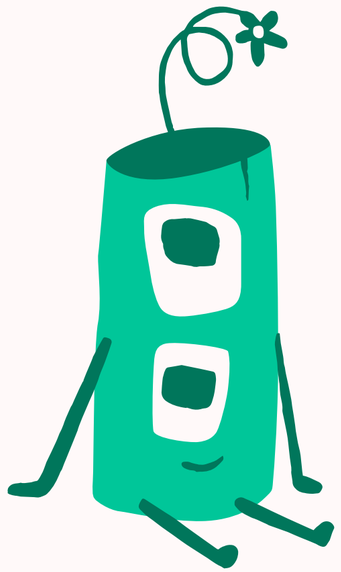
Condiciones

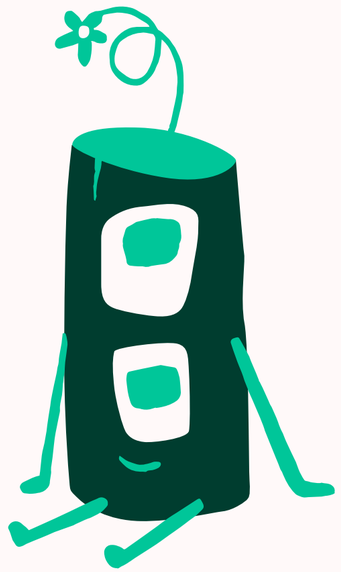
Aquí son tres preguntas distintas que se hacen, se verifica las tres veces si n es menor que el número indicado

```
def menor(n):
  if n<10:
    print('menor a 10')
  if n<20:
    print('menor a 20')
  if n<30:
    print('menor a 30')

menor(7)
```

```
menor a 10
menor a 20
menor a 30
```

Como se cumplen las tres condiciones, se imprime tres veces



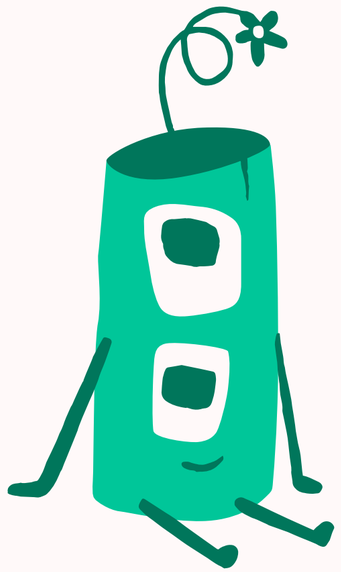


Condiciones

Aquí es una pregunta la que responde la función, si es un 0 o un 1, pero núnca será ambos así que no es necesario poner un elif

De todas formas si usted quiere puede poner elif y funciona igual

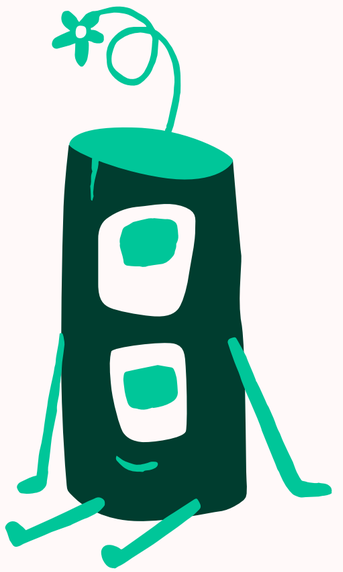
```
def unoOcero(n):
  if n==1:
    print('es uno')
  if n==0:
    print('es cero')
```



Recuerda que en este caso solo se imprime la respuesta, si defines a=unoOcero(1), a será del tipo None

```
a=unoOcero(1)
print(a)
```

```
es uno
None
```



AND / OR / NOT

and

Es verdadero solo cuando todas las condiciones que une son verdaderas

or

Es verdadero siempre que una de las condiciones que une sea verdadera

not

Es verdadero cuando la condición a la que antecede es falsa

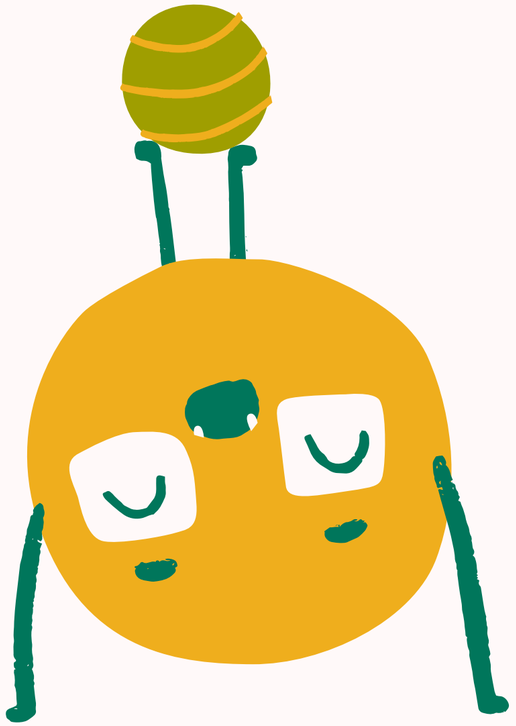
Cumplen con esta tablita

AND	VERDADERO	FALSO
VERDADERO	VERDADERO	FALSO
FALSO	FALSO	FALSO

OR	VERDADERO	FALSO
VERDADERO	VERDADERO	VERDADERO
FALSO	VERDADERO	FALSO

```
assert n==1 and n==2
assert n==1 or n==2
assert not n==1
```

- El primer assert nunca se va a cumplir, ya que n no puede ser 1 y 2 al mismo tiempo
- El segundo assert se cumplirá cuando n sea 1 o 2
- El ultimo assert se cumplira cuando n sea distinto de 1



RECURRENCIA

Definición

Acción de volver a ocurrir o aparecer una cosa con cierta frecuencia o de manera iterativa

Caso base

Es muy importante que se defina y tenga claro el caso base, de lo contrario se entra en un loop 'eterno'. Este es el caso en el que se vaa terminar la recurrencia

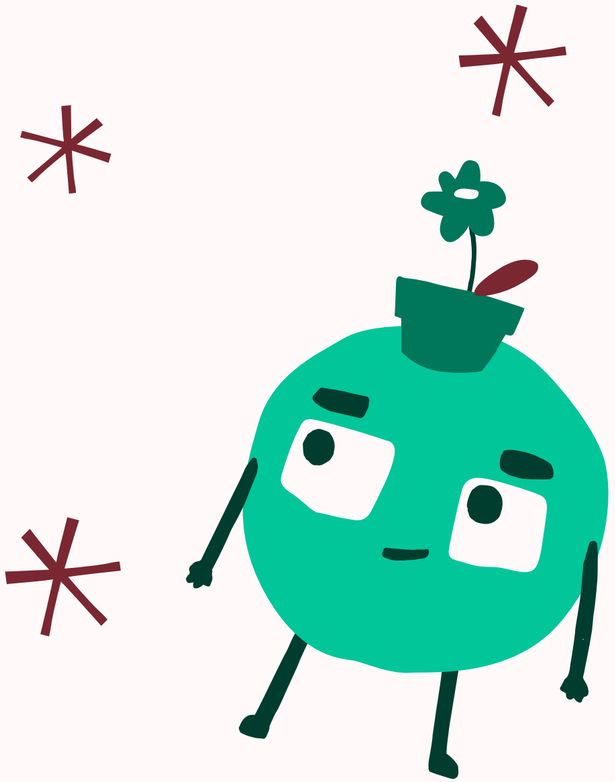
Caso recursivo

Es la sección donde definimos las acciones que se toman para hacer el problema más pequeño e implementar la recursividad en si

```
def saludo(n):
  print('Hola mundo')
  return saludo(n)

def saludo(n):
  print('Hola mundo')
  if n = 0:
    return
  else:
    return saludo(n-1)
```

No tiene un caso base, imprime hola mundo 'por siempre'

Tiene un caso base que le dice cuando detenerse y en el else esta como se hace el problema más pequeño



RECURRENCIA

Parametro auxiliar

Se puede definir un parámetro auxiliar al cual se le da un valor por default, en el caso de que no se le indique un valor en específico tomará el valor por default, ayuda mucho en la recurrencia

Se usa comúnmente cuando necesitamos guardar un valor a lo largo de la recurrencia

```python
#Cuenta la cantidad de unos que hay
#en un numero
def cuentaUnos(n,cantidad=0):
  if n%10 == 1:
    cantidad += 1
  if n <10:
    return cantidad
  else:
    return cuentaUnos(n//10,cantidad)
```

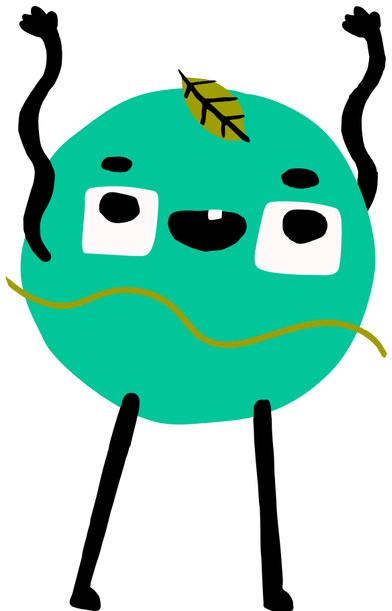
Al llamar a la función no es necesario escribir el parámetro cantidad, solo el n

Se edita el parámetro 'cantidad', que por default es 0, dentro de la función y luego se entrega como parámetro para guardar la cantidad de 1 dentro de la recursión

CORRER LAS FUNCIONES

Definir ≠ Correr

Al hacer un módulo interactivo deben recordar correr las funciones, no basta con definirlas con las mismas variables que le solicitaron al usuario previamente



Al definir únicamente la función, el programa no me imprime el saludo que yo quería

```
nombre = input('Escriba su nombre: ')

def saludo(nombre):
  print('Hola!' + nombre )
```

```
Escriba su nombre: Nadia
```

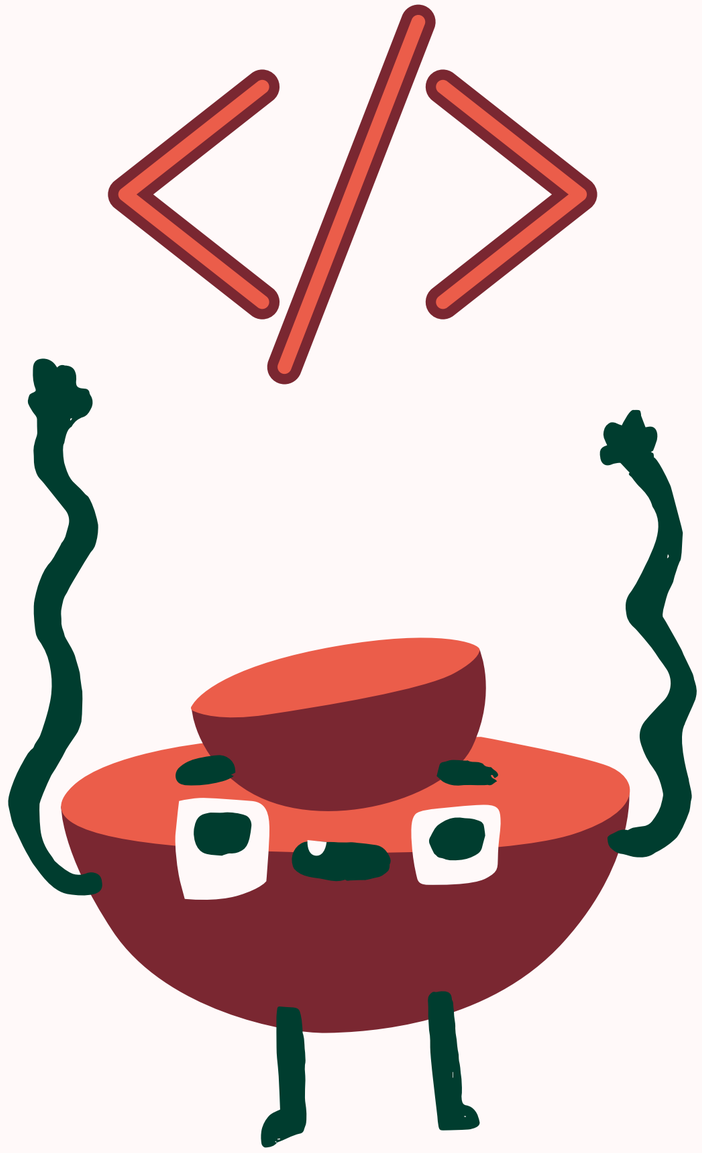
Es necesario llamar a la función con el parámetro para que se ejecute, no solo definirla

```
nombre = input('Escriba su nombre: ')

def saludo(nombre):
  print('Hola! ' + nombre )

saludo(nombre)
```

```
Escriba su nombre: Nadia
Hola! Nadia
```

TIPS PARA PROGRAMAR

CC-1002



PRINT CUANDO NO ENTIENDAS ALGO

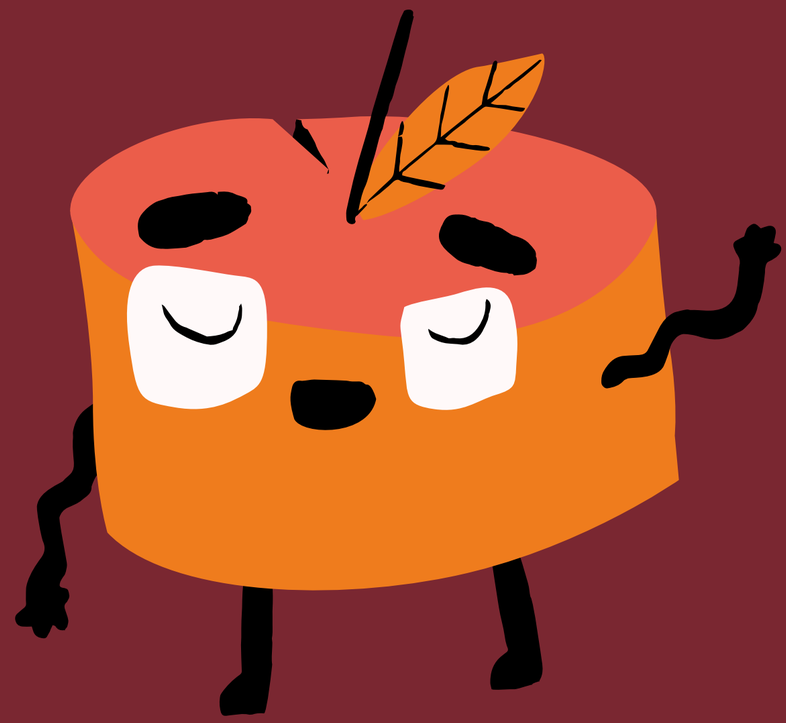
print("llego")

Si en algún momento no entienden que está pasando en su código o si está funcionando algo, escriban un print que les muestre que se está cumpliendo la condición que debería.

```
def esUno():
  n = input('ingresa un numero: ')
  if n == 1:
    return 'es uno'
  else: return 'no es uno'
esUno()
```

```
ingresa un numero: 1
'no es uno'
```

Supongamos que tengo esta función que me debe retornar 'es uno' en el caso de que el valor que se entrega es uno, como pueden ver, falla, y supongamos que no entiendo por que pasa esto



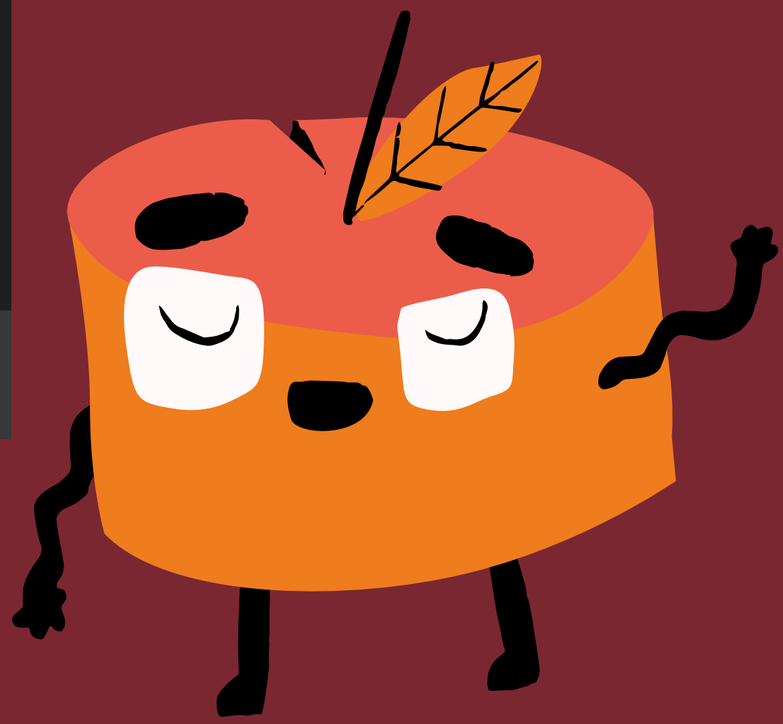
PRINT CUANDO NO ENTIENDAS ALGO

Agrego un print inmediatamente después de la condición que se debería estar cumpliendo

```
def esUno():
  n = input('ingresa un numero: ')
  if n == 1:
    print('llego')
    return 'es uno'
  else: return 'no es uno'
esUno()
```

```
ingresa un numero: 1
'no es uno'
```

No me imprime el 'llego' así que sé que no se está cumpliendo la condición y me enfoco en ver que está pasando que no se cumple



COMENTAR PARA NO PERDER EL HILO

#Comentar

Si encuentras que estás haciendo una función muy larga (o en general) puedes ir comentando partes del código para saber que es lo que se supone que pase en esa línea, también sirve para cuando estés haciendo el cuerpo de una función y tengas muchas condiciones.

```
#Queremo ver si es un float,
#un str o un int y trabajar
#con ello
def queEs(x):
  if type(x)==str:
    #Aquí se va a trabajar si es un str
  if type(x)==float:
    #Aquí se va a trabajar si es un float
  if type(x)==int:
    #Aqui se va a trabajar si es un int
```

Todavía no tengo definido lo que se va a hacer en cada sección, pero ya definí las condiciones y ahora puedo pasar a escribir el resto del código

BACKSLASH PARA DIVIDIR LÍNEAS \ LARGAS

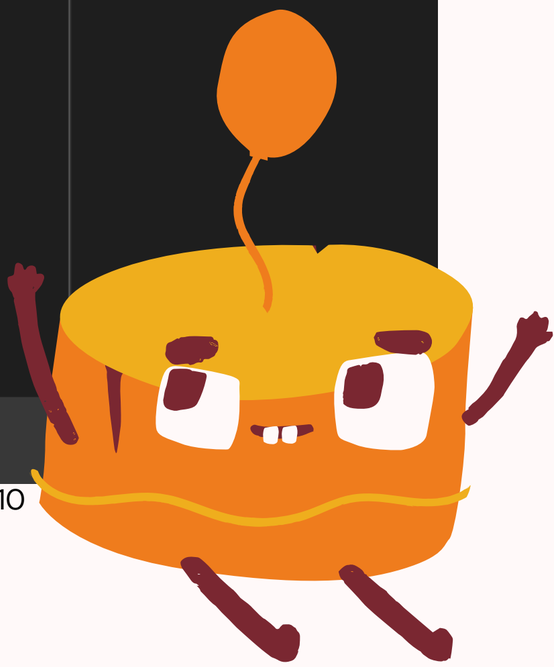
BACKSLASH \

Probablemente en algún momento tengan que programar algo en lo que una línea del código sea muy larga y la quieren poder visualizar completa en la pantalla, para eso pueden usar el símbolo \

```
def estaEntreUnoYDies(n):
  if (n==1) or (n==2) or (n==3) or (n==4) or (n==5) or (n==6) or (n==7) or (n==8) or (n==9) or (n==10):
    return 'si'
  else:
    return 'no'

def estaEntreUnoYDies2(n):
  if (n==1) or (n==2) or (n==3) or \
     (n==4) or (n==5) or (n==6) or \
     (n==7) or (n==8) or (n==9) or \
     (n==10):
     return 'si'
  else:
    return 'no'


print(estaEntreUnoYDies(5))
print(estaEntreUnoYDies2(5))
```

```
si
si
```

Esto es una exageración, hay mejores formas de ver si esta etre 1 y 10

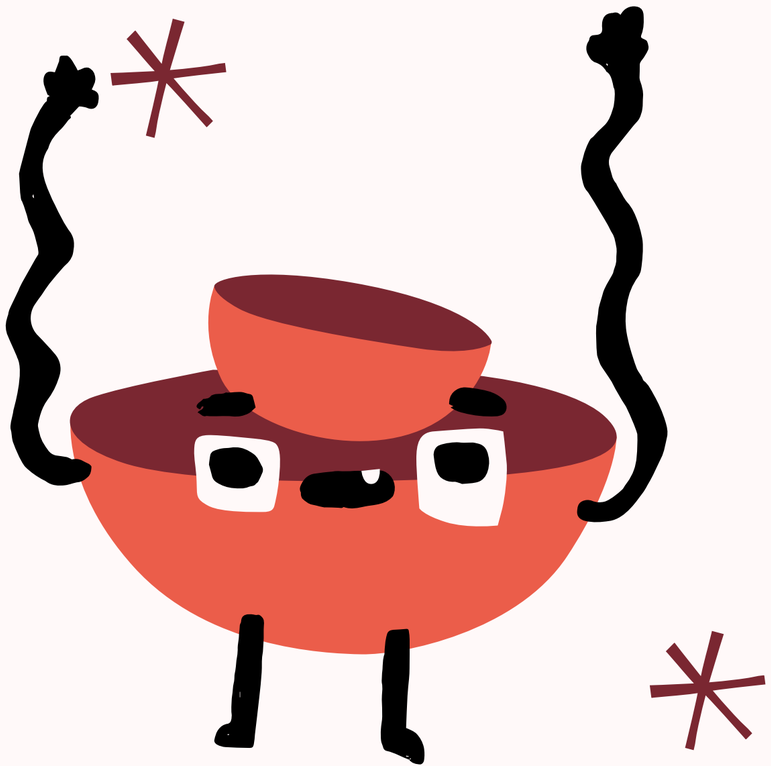


Ambas funciones hacen lo mismo y están escritas con las mismas condiciones, es más que nada por 'estética' y no es para nada obligatorio usarlo

RE DEFINIR VARIABLES

+= , -= y *=

Cuando se está programando muchas veces es necesario hacer cambios en una variable utilizando esta misma, como por ejemplo cuando queremos ir aumentando o disminuyendo un número en una recurrencia, para eso se pueden usar los comandos +=, -= o *=



Todas estas operaciones son equivalentes con las que tienen al lado, con la notación mostrada se ahorra el tener que volver a escribir la variable. Nuevamente esto no es algo obligatorio, pero algunas veces hace las cosas más fáciles

```
a = 12
a += 2
print(a)
```

```
14
```

```
a = 12
a = a + 2
print(a)
```

```
14
```

```
b = 12
b -= 2
print(b)
```

```
10
```

```
b = 12
b = b - 2
print(b)
```

```
10
```

```
c = 12
c = c * 2
print(c)
```

```
24
```

```
c = 12
c *= 2
print(c)
```

```
24
```