

NIVEL 1

FUNCIONES Y VARIABLES LOCALES



FUNCIONES

Además de las funciones de Python, que ya vimos, ¡podemos definir nuestras propias funciones!

- ✓ Es una forma de extender el lenguaje y de enseñarle a Python a hacer cálculos que inicialmente no sabe hacer

La sintaxis para definir nuestras propias funciones es:

```
def nombre( parámetros ):
    instrucciones
```

Denominaremos **invocar** o **llamar** a una función a la acción de usarla. Las funciones reciben cero, uno o más parámetros separados por comas y encerrados entre un par de paréntesis y pueden devolver un valor o no devolver nada.

```
In [1]: abs(-3)
Out[1]: 3

In [2]: abs(round(2.45,1))
Out[2]: 2.5
```

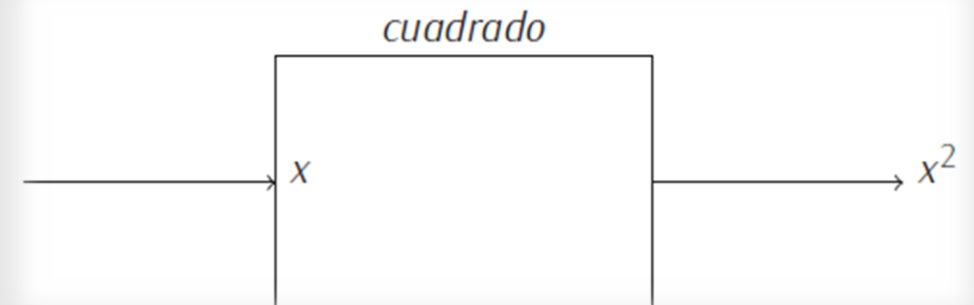
1 parámetro

2 parámetros

DEFINICIÓN DE FUNCIONES CON UN SOLO PARÁMETRO

Esta función recibe un número flotante y devuelve el cuadrado de dicho número

```
cuadrado.py*  
1  #!/usr/bin/env python3  
2  # -*- coding: utf-8 -*-  
3  
4  def cuadrado(x: float)->float:  
5      """  
6          Calcula el cuadrado de un número dado  
7      """  
8      return x ** 2  
9
```



PARTES DE UNA FUNCIÓN

Nombre de la función

La regla para poner los nombre es que funciones, variables y parámetros **NO** pueden tener el mismo nombre dentro de una misma función

Encabezado o signature

Empieza con la palabra reservada **def** y termina con **:**

Tipo del valor de retorno

```
def cuadrado(x: float) -> float:
```

```
    return x ** 2
```

Cuerpo de la función

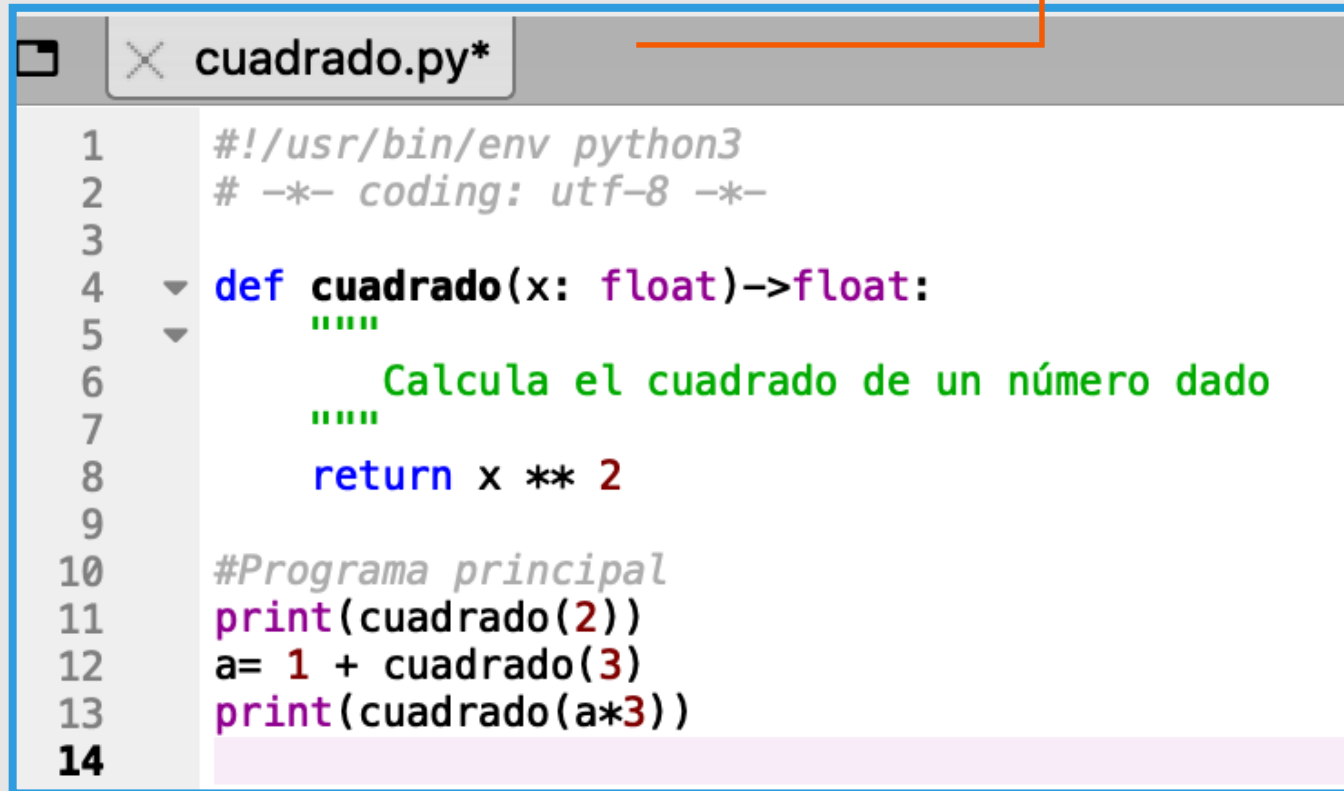
Conjunto de instrucciones que la componen (1 o más instrucciones). Lo que delimita el cuerpo de la función es la **INDENTACIÓN!**

Parámetro (valor de entrada) con su tipo

Instrucción de retorno Se usa para devolver un valor

PROGRAMA COMPLETO

Programa cuadrado.py



```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  def cuadrado(x: float)->float:
5      """
6          Calcula el cuadrado de un número dado
7      """
8      return x ** 2
9
10 #Programa principal
11 print(cuadrado(2))
12 a= 1 + cuadrado(3)
13 print(cuadrado(a*3))
14
```

PROGRAMA COMPLETO

Programa cuadrado.py

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  def cuadrado(x: float)->float:
5      """
6      Calcula el cuadrado de un número dado
7      """
8      return x ** 2
9
10 #Programa principal
11 print(cuadrado(2))
12 a= 1 + cuadrado(3)
13 print(cuadrado(a*3))
14
```

Definición de la función llamada cuadrado

PROGRAMA COMPLETO

```
cuadrado.py*
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  def cuadrado(x: float)->float:
5      """
6      Calcula el cuadrado de un número dado
7      """
8      return x ** 2
9
10 #Programa principal
11 print(cuadrado(2))
12 a= 1 + cuadrado(3)
13 print(cuadrado(a*3))
14
```

Programa cuadrado.py

Definición de la función llamada cuadrado

Programa principal: instrucciones que no son cuerpo de funciones y que se ejecutan cuando el programa entra en acción

PROGRAMA COMPLETO

```
cuadrado.py*
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  def cuadrado(x: float)->float:
5      """
6      Calcula el cuadrado de un número dado
7      """
8      return x ** 2
9
10 #Programa principal
11 print(cuadrado(2))
12 a= 1 + cuadrado(3)
13 print(cuadrado(a*3))
14
```

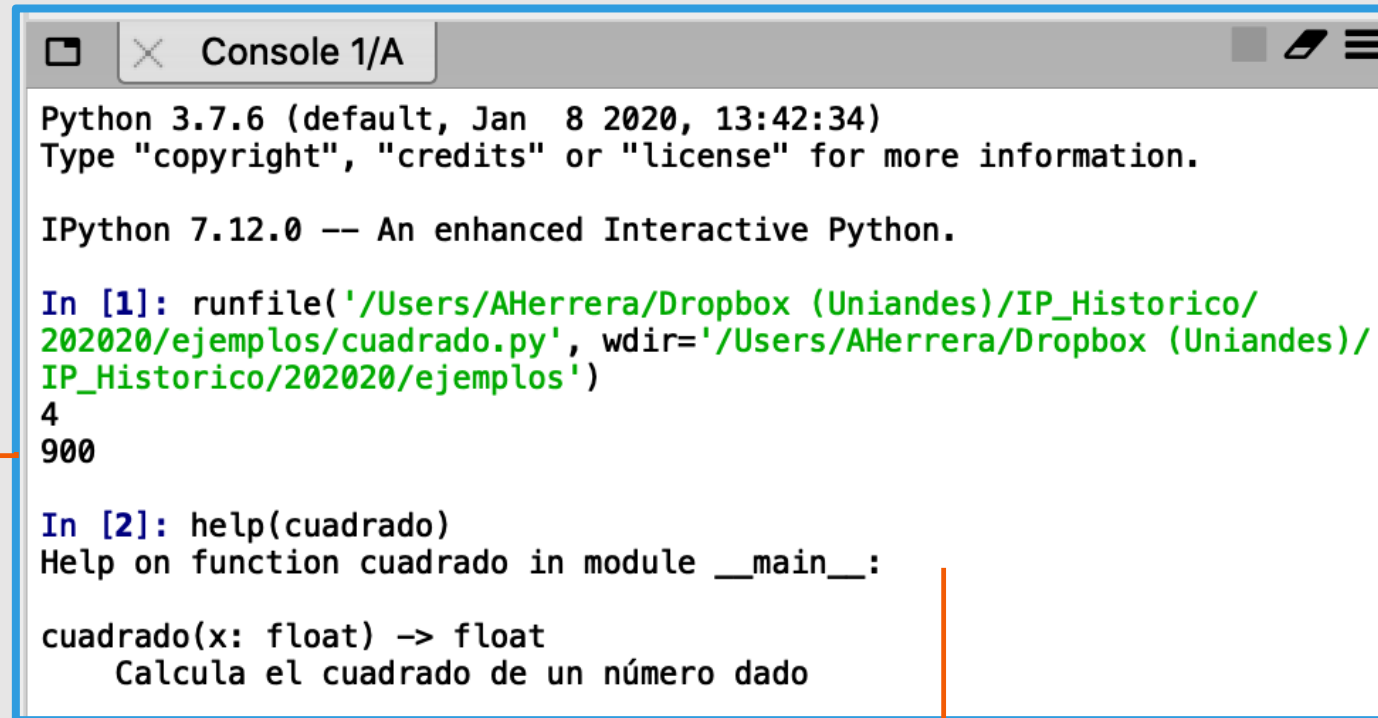
Programa cuadrado.py

Diferentes formas de incluir comentarios

Argumento es el valor que damos al parámetro cuando llamamos la función

PROGRAMA COMPLETO

• Resultado de la ejecución



```
Python 3.7.6 (default, Jan 8 2020, 13:42:34)
Type "copyright", "credits" or "license" for more information.

IPython 7.12.0 -- An enhanced Interactive Python.

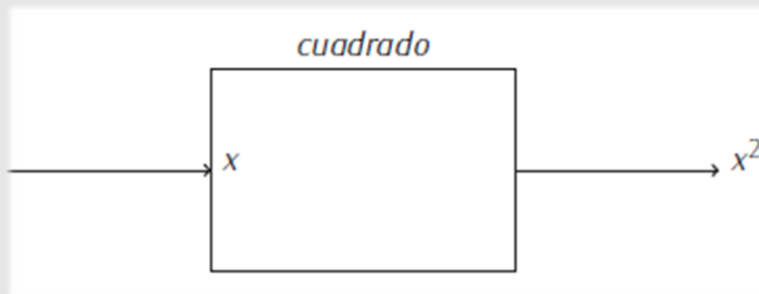
In [1]: runfile('/Users/AHerrera/Dropbox (Uniandes)/IP_Historico/
202020/ejemplos/cuadrado.py', wdir='/Users/AHerrera/Dropbox (Uniandes)/
IP_Historico/202020/ejemplos')
4
900

In [2]: help(cuadrado)
Help on function cuadrado in module __main__:

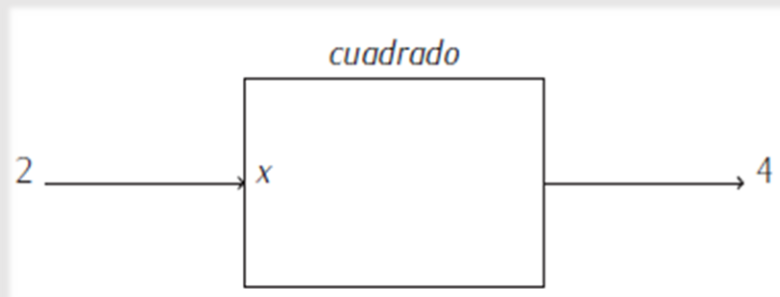
cuadrado(x: float) -> float
    Calcula el cuadrado de un número dado
```

• Resultado de la función help

DEFINIR ES DIFERENTE DE INVOCAR



Definir es “crear” la caja que calcula el cuadrado de un número cualquiera



Invocar o llamar es “usar” la caja para calcular el cuadrado de un número específico que se pasa como argumento

DEFINIR VS. INVOCAR

```
cuadrado.py ✕  
1 # -*- coding: utf-8 -*-  
2  
3  
4 def cuadrado(x: float)->float:  
5     return x ** 2  
6
```

Si solo definimos una función y no la usamos (llamamos o invocamos), no pasa nada en ejecución

Resultado de la ejecución

```
In [9]: runfile('D:/Marcela/UniAndes/Cursos/IP/201
```

```
In [10]:
```

```
cuadrado.py ✕  
1 # -*- coding: utf-8 -*-  
2  
3  
4 def cuadrado(x: float)->float:  
5     return x ** 2  
6  
7  
8 print(cuadrado(2))  
9 a = 1 + cuadrado(3)  
10 print(cuadrado(a * 3))  
11
```

Hay que usar la función e imprimir el valor de retorno para que se vea en pantalla



A continuación vamos a
hablar de las confusiones
más comunes cuando
usamos funciones

NO CONFUNDIR...



Funciones diferentes son independientes

```
def cuadrado(x: float)->float:
    return x ** 2

def cubo(x: float)->float:
    return x * x * x

print(cuadrado(2))
print(cubo(3))
```

Las dos funciones reciben un parámetro cada una. Pueden tener el mismo nombre, son parámetros DIFERENTES porque están en funciones diferentes

Resultado de la ejecución

```
In [10]: runfile('D:/Marcela/Unia
4
27
```

Parámetro es diferente a argumento

```
def cubo(x: float)->float:
    return x * x * x

y = 2
print(cubo(y))
```

y es una variable que se usa para guardar el valor 2

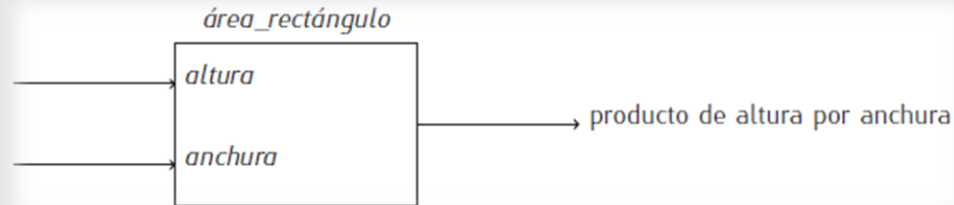
El parámetro de la función es **x**, pero el argumento con el que se invoca es el valor de la variable **y**. Lo que importa en el llamado es el **VALOR**.

Resultado de la ejecución

```
In [11]: runfile('D:/Marcela/
8
```

DEFINICIÓN DE FUNCIONES CON VARIOS PARÁMETROS

Esta función devuelve el valor del área de un rectángulo dadas su altura y su anchura



```
def area_rectangulo(altura: float, anchura: float)->float:
    return altura * anchura

def area_rectangulo_2(altura: float, anchura: float)->float:
    area = altura * anchura
    return area
```

Los diferentes parámetros se separan con coma

DEFINICIÓN Y USO DE FUNCIONES CON VARIOS PARÁMETROS

Al usar la función, los argumentos también deben separarse por comas

```

1
2
3 def area_rectangulo(altura: float, anchura: float)->float:
4     return altura * anchura
5
6
7 x = float(input("Digite la altura: "))
8 y = float(input("Digite la anchura: "))
9 area = area_rectangulo(x, y)
10 print("El área del rectángulo es: ", area)
11

```

Resultado de la ejecución

```

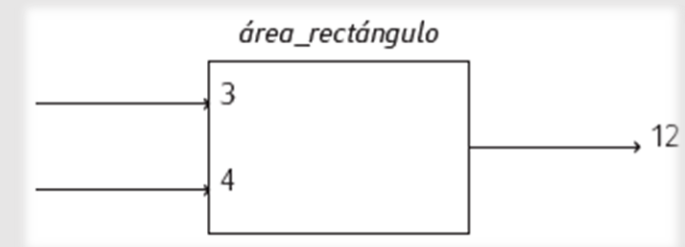
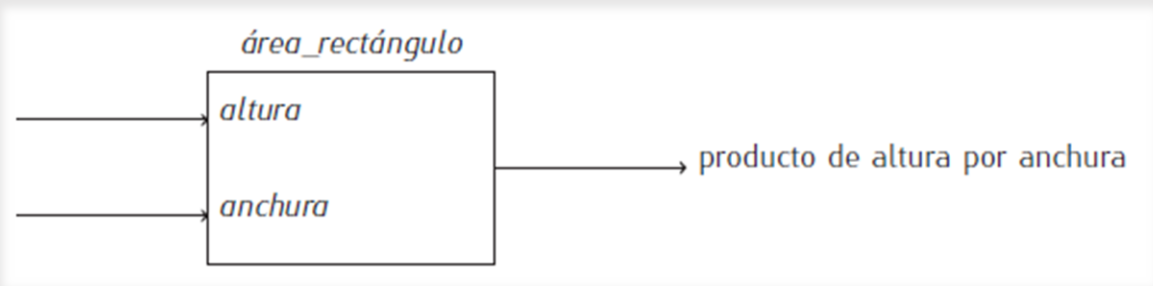
Terminal de IPython
Terminal 1/A

In [12]: runfile('C:/Users/Mhernand

Digite la altura: 3

Digite la anchura: 4
El área del rectángulo es: 12.0

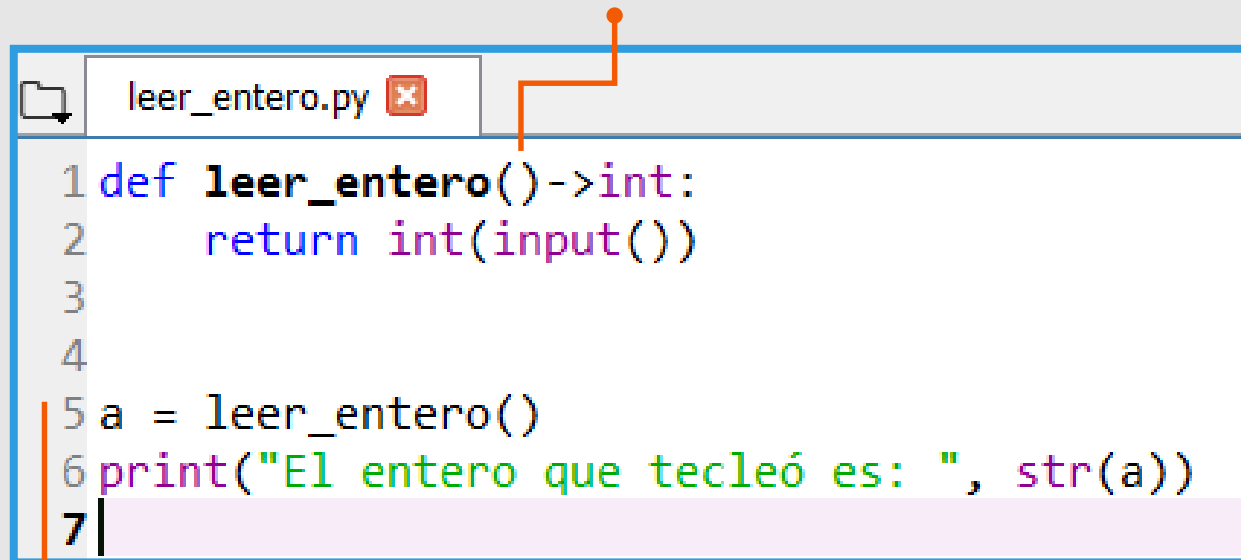
```



El orden de los argumentos debe ser el MISMO de los parámetros: **x** «reemplaza» el parámetro altura, **y** el parámetro anchura

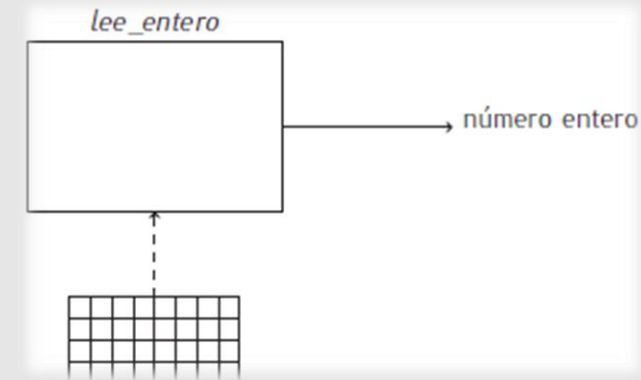
DEFINICIÓN DE FUNCIONES SIN PARÁMETROS

Esta función lee un valor del teclado (con `input`), pero no tiene ningún parámetro

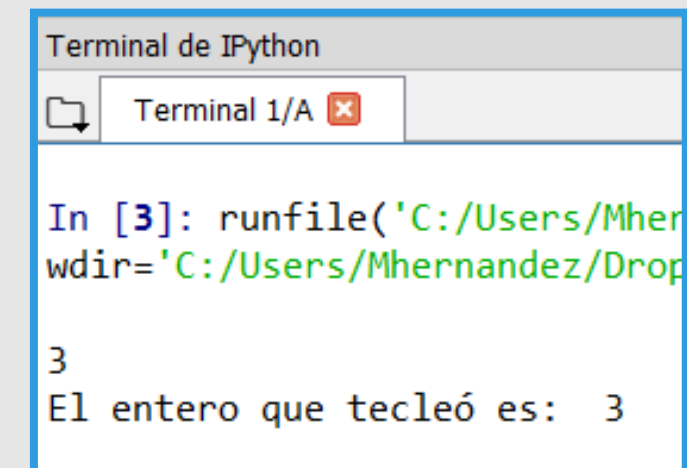


```
1 def leer_entero()->int:
2     return int(input())
3
4
5 a = leer_entero()
6 print("El entero que tecleó es: ", str(a))
7 |
```

Cuando se invoca, hay que poner los paréntesis, aunque no haya argumentos



Resultado de la ejecución



```
Terminal de IPython
Terminal 1/A
In [3]: runfile('C:/Users/Mhernandez/Drop
wdir='C:/Users/Mhernandez/Drop
3
El entero que tecleó es: 3
```


¿PARÁMETROS O TECLADO?



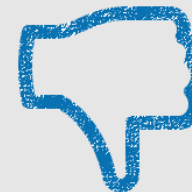
- Un error frecuente al diseñar funciones consiste en tratar de obtener la información directamente del teclado
- No es que esté prohibido, pero es excepcional que una función obtenga la información de ese modo.
- Cuando nos pidan diseñar una función que recibe uno o más datos, se sobreentiende que son parámetros

```
cubo_correcto.py
1 def cubo(x: int)->int:
2     return x * x * x
3
4
5 valor = int(input("Teclee el valor: "))
6 print("El resultado es: ", str(cubo(valor)))
7
```



```
Terminal de IPython
Terminal 1/A
In [5]: runfile('C:/User
Teclee el valor: 4
El resultado es: 64
```

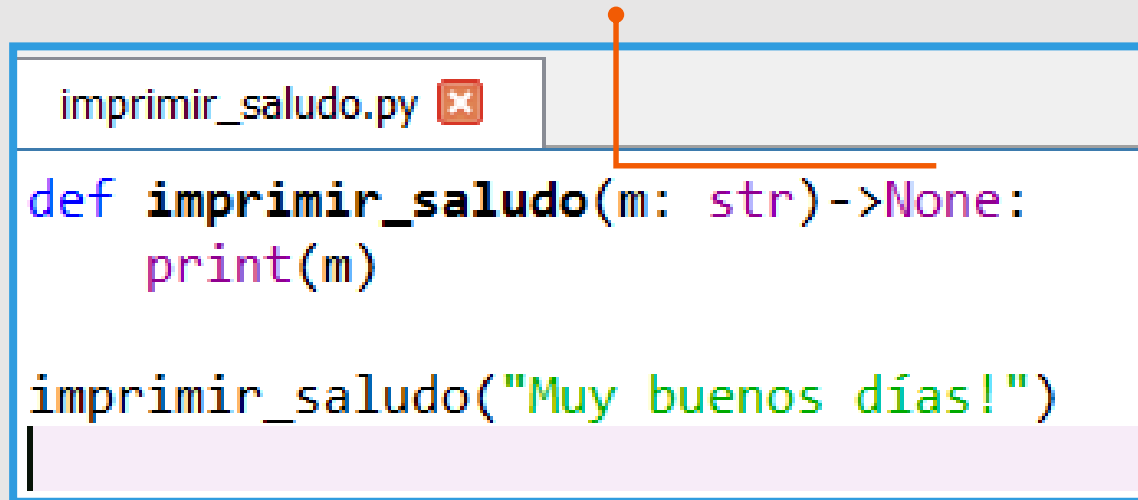
```
cubo_incorrecto.py
1 def cubo()->int:
2     x = int(input("Teclee el valor: "))
3     return x * x * x
4
5
6 print("El resultado es: ", str(cubo()))
7
```



```
Terminal de IPython
Terminal 1/A
In [7]: runfile('C:/User
Teclee el valor: 4
El resultado es: 64
```

FUNCIONES QUE NO DEVUELVEN NINGÚN VALOR

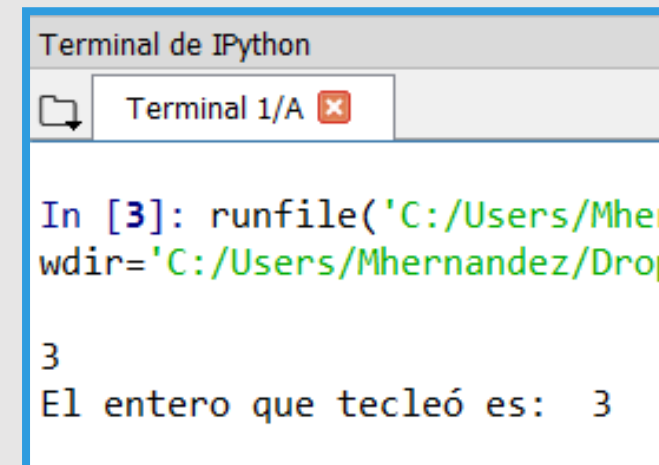
Este tipo de funciones, también conocidas como procedimientos, imprimen por pantalla



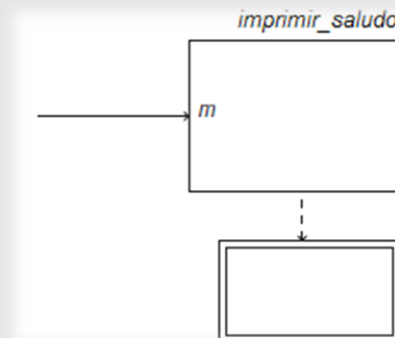
```
imprimir_saludo.py ✕  
  
def imprimir_saludo(m: str) -> None:  
    print(m)  
  
imprimir_saludo("Muy buenos días!")  
|
```

Este procedimiento NO devuelve (retorna) ningún valor

Resultado de la ejecución



```
Terminal de IPython  
Terminal 1/A ✕  
  
In [3]: runfile('C:/Users/Mher  
wdir='C:/Users/Mhernandez/Drop  
  
3  
El entero que tecleó es: 3
```



¿VALOR DE RETORNO O PANTALLA?



- Hemos visto que es posible imprimir información directamente por pantalla desde una función. Esto solo lo hacemos cuando el propósito de la función es mostrar esa información
- Es un error sustituir la instrucción **return** por la instrucción **print**
- Cuando nos pidan diseñar una función que devuelva un valor, debe hacerse con la instrucción **return**
- Mostrar algo por pantalla no es devolver ese algo



```
def ultima_cifra(numero: int)->int:
    return(numero % 10)

x = int(input("Digite el número: "));
print("El último dígito de ",str(x),"es: ",str(ultima_cifra(x)))
```

Resultado de la ejecución

```
In [1]: runfile('D:/Dropbox/Dropbox (Uniandes)/R
Nivel 1/4-C3/EjemplosPyDiapositivas')
```

```
Digite el número: 4567
El último dígito de 4567 es: 7
```



```
ultima_cifra_incorrecto.py x
def ultima_cifra(numero: int)->None:
    print("El último dígito de",numero,"es :", str(numero % 10))

x = int(input("Digite el número: "))
ultima_cifra(x)
```

Resultado de la ejecución

```
Terminal de IPython
Terminal 1/A x

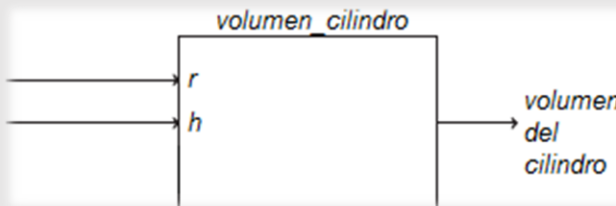
In [8]: runfile('D:/Dropbox/Dropbox (Uniandes)/R
curso/Piloto 201910/Nivel 1/4-C3/E

Digite el número: 4567
El último dígito de 4567 es : 7
```



VARIABLES LOCALES

- ✓ En el cuerpo de las funciones es posible definir y usar variables
- ✓ Se diferencian de las variables que definimos fuera de cualquier función, es decir, en lo que llamamos el **programa principal**



VARIABLES LOCALES

`area_base` es una variable local de la función `volumen_cilindro`

```
volumen_cilindro.py ✕  
1 def volumen_cilindro(radio_base: float, altura: float) -> float:  
2     area_base = 3.1416 * radio_base * radio_base  
3     return area_base * altura  
4  
5  
6 r = float(input("Digite el radio: "))  
7 a = float(input("Digite la altura: "))  
8 print("El volumen es ", str(volumen_cilindro(r, a)))
```

Resultado de la ejecución

```
Terminal de IPython  
Terminal 1/A ✕  
  
In [29]: runfile('C:/Users/Mhernan  
  
Digite el radio: 3.5  
  
Digite la altura: 4.24  
El volumen es 163.17470400000002
```

VARIABLES LOCALES



```

volumen_cilindro.py
1 def volumen_cilindro(radio_base: float, altura: float)->float:
2     area_base = 3.1416 * radio_base * radio_base
3     return area_base * altura
4
5
6 r = float(input("Digite el radio: "))
7 a = float(input("Digite la altura: "))
8 print("El volumen es ", str(volumen_cilindro(r, a)))
9 print(str(area_base))

```

area_base solo existe en el cuerpo de la función. Fuera del cuerpo, no está definida y no se puede usar

Resultado de la ejecución (ERROR)

```

Terminal de IPython
Terminal 1/A

In [29]: runfile('C:/Users/Mhernandez/Desktop/IP/volumen_cilindro.py')

Digite el radio: 3.5

Digite la altura: 4.24
El volumen es 163.17470400000002
Traceback (most recent call last):

  File "<ipython-input-29-8a055f51a660>", line 9, in <module>
    runfile('C:/Users/Mhernandez/Desktop/IP/volumen_cilindro.py')

  File "C:\Anaconda3\lib\site-packages\spyder_kernels\console\runfile.py", line 112, in runfile
    execfile(filename, namespace)

  File "C:\Anaconda3\lib\site-packages\spyder_kernels\console\runfile.py", line 112, in execfile
    exec(compile(f.read(), filename, 'exec'), namespace)

  File "C:/Users/Mhernandez/Desktop/IP/volumen_cilindro.py", line 9, in <module>
    print(str(area_base))

NameError: name 'area_base' is not defined

```

VARIABLES LOCALES

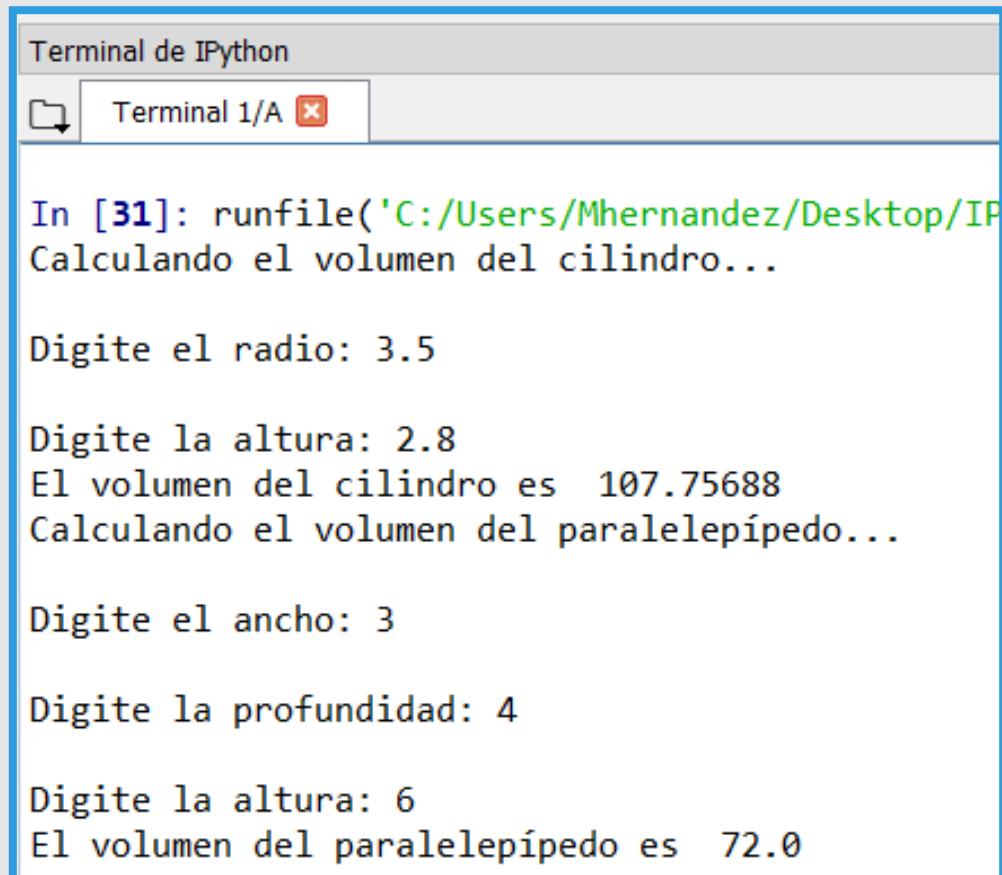
Como las variables locales solo existen dentro de las funciones, puede haber variables con el mismo nombre en funciones diferentes y son variables diferentes



```
volumen_cilindro_paralelepipedo.py x
1 def volumen_cilindro(radio_base: float, altura: float)->float:
2     area_base = 3.1416 * radio_base * radio_base
3     return area_base * altura
4
5
6 def volumen_paralelepipedo(a: float, b: float, c: float)->float:
7     area_base = a * b
8     return area_base * c
9
10
11 print("Calculando el volumen del cilindro...")
12 r = float(input("Digite el radio: "))
13 a = float(input("Digite la altura: "))
14 print("El volumen del cilindro es ", str(volumen_cilindro(r, a)))
15
16 print("Calculando el volumen del paralelepípedo...")
17 ancho = float(input("Digite el ancho: "))
18 profundidad = float(input("Digite la profundidad: "))
19 altura = float(input("Digite la altura: "))
20 print("El volumen del paralelepípedo es ",
21       str(volumen_paralelepipedo(ancho, profundidad, altura)))
22
```

VARIABLES LOCALES

Resultado de la ejecución



```
Terminal de IPython
Terminal 1/A x

In [31]: runfile('C:/Users/Mhernandez/Desktop/IP...
Calculando el volumen del cilindro...

Digite el radio: 3.5

Digite la altura: 2.8
El volumen del cilindro es  107.75688
Calculando el volumen del paralelepípedo...

Digite el ancho: 3

Digite la profundidad: 4

Digite la altura: 6
El volumen del paralelepípedo es  72.0
```


LOS PARÁMETROS NO SE MODIFICAN (PARÁMETROS POR VALOR)

Aunque se modifique el valor de un parámetro ...

...cuando la función termina, el argumento conserva su valor original

```
parametros.py
1 def incrementar(a: int) -> int:
2     a = a + 1
3     return a
4
5
6 a = 1
7 b = incrementar(a)
8
9 print("Valor de a: ", a)
10 print("Valor de b: ", b)
```

```
Terminal de IPython
Terminal 2/A

In [2]: runfile('D:/
wdir='D:/Dropbox/Drc
Valor de a:  1
Valor de b:  2
```