Funciones en Python

¿Por qué necesitamos funciones?

- Hasta ahora hemos encontrado algunas funciones pero las hemos usado como herramientas para facilitar la vida y simplificar las tareas que consumen mucho tiempo.
- Ejemplos:
- Cuando se desee que se impriman algunos datos en la consola, hemos usado print().
- Cuando desee leer el valor de una variable, hemos utilizado input(), junto con int() o float().

- A menudo sucede que un fragmento de código en particular se **repite muchas veces en su programa**.
- Se repite literalmente o con solo algunas modificaciones menores, que consisten en el uso de otras variables en el mismo algoritmo.
- También sucede que un programador no puede resistirse a simplificar el trabajo, y comienza a clonar dichos fragmentos de código utilizando el portapapeles y las operaciones de copiar y pegar.

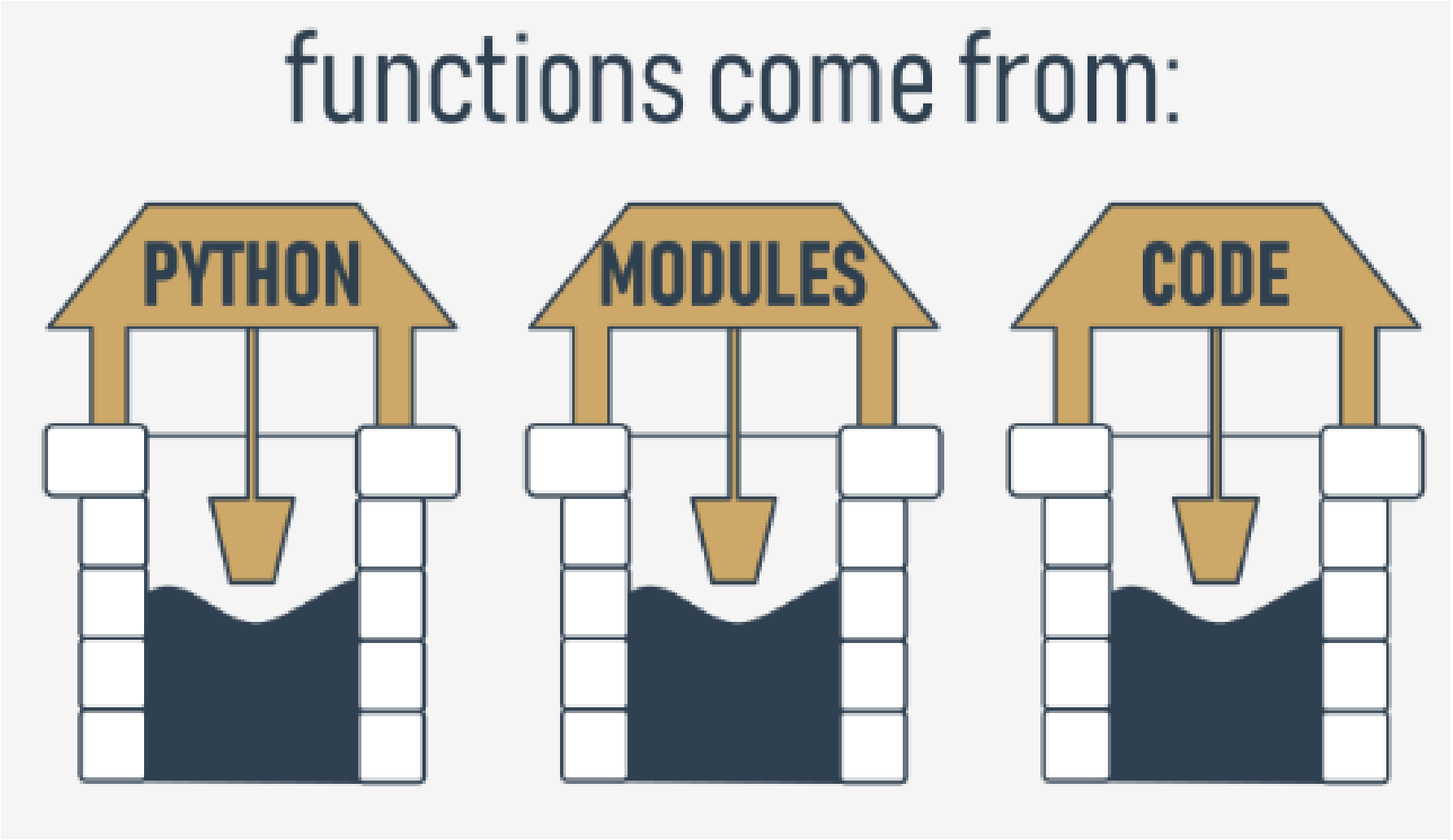
```
print("Enter a value: ")
    a = int(input())

print("Enter a value: ")
    b = int(input())

print("Enter a value: ")
    c = int(input())
```

Que es una función?

- Una **función** es un bloque de código que realiza una tarea específica cuando se llama a la función (se invoca).
- Puede usar funciones para hacer que su código sea reutilizable, mejor organizado y más legible.
- Las funciones pueden tener parámetros y valores de retorno.


functions come from:
PYTHON
MODULES
CODE

¿De dónde vienen las funciones?

- Hay al menos cuatro tipos básicos de funciones en Python
- Desde el propio Python: numerosas funciones (como print()) son una parte integral de Python y siempre están disponibles a estas funciones se las conoce como **funciones integradas** ;
- De los módulos preinstalados de Python : muchas funciones, muy útiles, pero que se utilizan con mucha menos frecuencia que las integradas, están disponibles en varios módulos instalados junto con Python; el uso de estas funciones requiere algunos pasos adicionales del programador para que sean completamente accesibles.
- Directamente desde su código : puede escribir sus propias funciones, colocarlas dentro de su código y usarlas libremente;
- Hay otra posibilidad, pero está relacionada con las clases.

NOTAS:

- **No debe invocar una función que no se conoce en el momento de la invocación.**

```
print("We start here.")
message()
print("We end here.")

def message():
    print("Enter a value: ")
```

- **No debe tener una función y una variable del mismo nombre.**

```
def message():
    print("Enter a value: ")

message = 1
```

Ejemplo:

```
def message():
    print("Enter a value: ")

message()
a = int(input())
message()
b = int(input())
message()
c = int(input())
```

Modificar el mensaje de solicitud ahora es fácil y claro: puede hacerlo cambiando el código en un solo lugar , dentro del cuerpo de la función.

Funciones con Parámetros

- Puede pasar información a las funciones mediante el uso de parámetros.
- Sus funciones pueden tener tantos parámetros como necesite.

```
def hi(name):
    print("Hi,", name)


hi("Greg")
```

Ejemplos:

Función de dos parámetros:

```
def hiAll(name1, name2):
    print("Hi,", name2)
    print("Hi,", name1)

hiAll("Sebastian", "Konrad")
```

- **Función de tres parámetros:**

```
def address(street, city, postalCode):
    print("Your address is:", street, "St.,", city, postalCode)

s = input("Street: ")
pC = input("Postal Code: ")
c = input("City: ")

address(s, c, pC)
```

Formas de pasar argumentos

```
Ex. 1
def subtra(a, b):
    print(a - b)

subtra(5, 2)     # outputs: 3
subtra(2, 5)     # outputs: -3
```

```
Ex. 2
def subtra(a, b):
    print(a - b)

subtra(a=5, b=2)     # outputs: 3
subtra(b=2, a=5)     # outputs: 3
```

```
Ex. 3
def subtra(a, b):
    print(a - b)

subtra(5, b=2)     # outputs: 3
subtra(5, 2)     # outputs: 3
```

Retorno de valores en funciones.

- Puede usar la palabra clave return para indicarle a una función que devuelva algún valor.
- La declaración return sale de la función, por ejemplo:

```
def multiply(a, b):
    return a * b

print(multiply(3, 4))     # outputs: 12

def multiply(a, b):
    return

print(multiply(3, 4))     # outputs: None
```

El resultado de una función puede asignarse fácilmente a una variable, por ejemplo:

```
def wishes():
    return "Happy Birthday!"

w = wishes()

print(w)     # outputs: Happy Birthday!
```

Mire la diferencia en la salida en los siguientes dos ejemplos:

```
# Example 1
def wishes():
    print("My Wishes")
    return "Happy Birthday"

wishes()    # outputs: My Wishes


# Example 2
def wishes():
    print("My Wishes")
    return "Happy Birthday"

print(wishes())    # outputs: My Wishes
                   #          Happy Birthday
```

Puede usar una lista como argumento de una función, por ejemplo:

```
def hiEverybody(myList):
    for name in myList:
        print("Hi,", name)

hiEverybody(["Adam", "John", "Lucy"])
```

Una lista también puede ser el resultado de una función, por ejemplo:

```
def createList(n):
    myList = []
    for i in range(n):
        myList.append(i)
    return myList


print(createList(5))
```

Funciones Lambda

En Python

Funciones lambda

- Se trata de crear funciones de manera rápida, just in time, sobre la marcha, para prototipos ligeros que requieren únicamente de una pequeña operación o comprobación.
- Por lo tanto, toda función lambda también puede expresarse como una convencional (pero no viceversa).

```
def a(x, y):
    return x + y

b = lambda x, y: x + y
```

Funciones lambda

```
def a(x, y):
    return x + y

b = lambda x, y: x + y
```

- En la imagen se observa cómo está constituida una función lambda y en comparación a una convencional.
- En color verde se remarcan los argumentos, y en rojo el valor de retorno.
- En base a esto, se conforma de la siguiente manera:

Funciones lambda

lambda argumentos: resultado

O bien se puede asignar la función a una variable:

f = lambda argumentos: resultado

- Y en función normal seria:

```
def f(argumentos):
        return resultado
```

Funciones lambda

La razón por la cual querrás emplear este tipo de funciones es simplemente por rapidez (durante el desarrollo, no la ejecución) y, en ciertos casos, claridad del código.

Ejemplo:

```
>>> a = [0, 1, -1, -2, 3, -4, 5, 6, 7]
>>> b = filter(lambda x: x > 0, a)
>>> b
[1, 3, 5, 6, 7]
```

```
def f(x):
    return x > 0
b = filter(f, a)
```