

Esta clase va a ser

- grabada

a

Clase 09. PYTHON

Funciones II

Temario

08

Funciones I

- ✓ Funciones
- ✓ Retornando valores
- ✓ Enviando valores

09

Funciones II

- ✓ [Argumentos y parámetros](#)

10

Excepciones

- ✓ Errores y Excepciones

Objetivos de la clase



Reconocer los tipos de argumentos y parámetros.



Aplicar funciones recursivas e integradas.

Repositorio Github

Te dejamos el acceso al Repositorio de Github donde encontrarás todo el material complementario y scripts de la clase.

✓ [Repositorio Python](#)



Argumentos y parámetros



Para pensar

¿Cuál es la diferencia entre los parámetros y argumentos?

Contesta mediante el chat de Zoom

Argumentos y parámetros

Como sabemos, durante la definición de la función, las variables o valores se denominan parámetros:

```
def suma(numero1, numero2):  
    return numero1 + numero2
```

Y durante la llamada se le denominan argumentos, como los argumentos de los scripts.

```
resultado = suma(7, 5)
```

En esta clase estaremos viendo los distintos tipos de argumentos y parámetros.

Argumento por posición



Argumento por posición

Cuando se envían argumentos a una función, se reciben por orden en los parámetros definidos:

```
def suma(numero1, numero2):  
    return numero1 + numero2  
resultado = suma(7, 5)
```

El argumento 7 es la posición 0, por consiguiente es el parámetro de la función numero1, seguidamente el argumento 5 es la posición 1 por consiguiente es el parámetro de la función numero2.





Argumentos por posición

Si tomamos el siguiente ejemplo sabremos que la resta nos dará 3:

```
def resta(a, b):  
    return a - b  
resultado = resta(15, 12)  
resultado
```

Pero, si modificamos el orden de los argumentos nos dará otro resultado:

```
resultado = resta(12, 15)
```

Argumento por nombre



Argumentos por nombre

Como vimos, si pasamos ordenado el argumento, se verá reflejado ordenadamente el parámetro.

Para cambiar esto se utiliza la asignación de argumentos por nombre, si indicamos durante la llamada que valor tiene cada parámetro a partir de su nombre:

```
def resta(a, b):  
    return a - b  
resultado = resta(b=15, a=12)  
resultado
```



Argumentos por nombre

Recordemos que al utilizar argumentos por nombre, no importa el orden:

```
def resta(a, b, c):  
    return a - b - c  
resultado = resta(a=15, b=12, c=2)  
resultado
```



```
>>> resultado = resta(c=2, a=15,  
b=12)
```

Llamada sin argumentos



Llamada sin argumentos

Veamos qué pasa si llamamos una función con parámetros ya definidos:

```
def resta(a, b):  
    return a - b  
resultado = resta()
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: resta() takes exactly 2 arguments (0 given)

¿Cómo solucionamos el TypeError al momento de llamar una función sin argumento?



Parámetros por defecto

```
def resta(a=10, b=5):  
    return a - b  
resultado = resta()  
  
>>>5
```

Python, nos deja asignar unos valores por defecto a los parámetros, es decir, indicarle que tendrán un valor por defecto si no viene ningún valor.

Argumento por valor y referencia

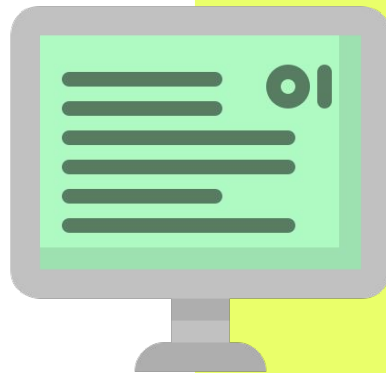
Argumentos

Si hablamos de argumentos tenemos que tener algo en cuenta:

Cuando **enviamos información a una función** generalmente **estos datos se envían por valor**.

Eso significa que se crea una copia dentro de la función de los valores que enviamos en sus propias variables.

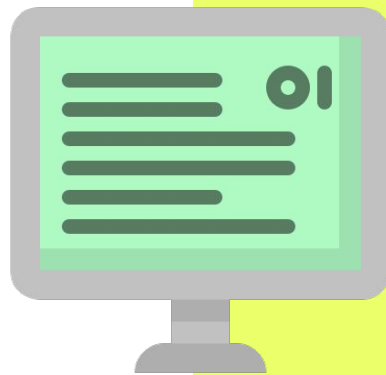
Pero, **hay casos excepcionales**, las colecciones, listas, diccionarios, conjuntos. Estos datos **se envían por referencia**.



Referencia

¿Que significa que los conjuntos como listas, tuplas, etc, se envíen por referencia?

Significa que, en lugar de una copia dentro de la función, estaremos manejando el dato original, y si lo modificamos también se verá reflejado en el exterior, es decir, en el conjunto original y no en una copia en la función. Esto debido a que hacen referencia a la variable externa, algo así como un acceso directo.





Dependiendo del tipo de dato:

Paso por valor: se crea una copia local de la variable dentro de la función.

Los tipos simples se pasan por valor:

Enteros, flotantes, cadenas, lógicos...

Paso por referencia: Se maneja directamente la variable, los cambios realizados dentro de la función le afectarán también fuera.

Los tipos compuestos se pasan por referencia:

Listas, diccionarios,
tuplas, conjuntos...



Paso por valor

Los números se pasan por valor y crean una copia dentro de la función, **no les afecta externamente** lo que hagamos con ellos en la función:

```
def doblar_valor(numero):  
    numero *= 2  
  
numero = 10  
doblar_valor(numero)  
print(numero)  
  
>>> 10
```

Fuente: [Hektor Docs](#)



Paso por referencia

```
» #Paso por referencia
def doblar_valores(numeros):
    for i,n in enumerate(numeros):
        numeros[i] *= 2
```

```
» listaDeNico = [10,50,100]
```

```
» doblar_valores(listaDeNico)
```

```
» listaDeNico
[20, 100, 200]
```

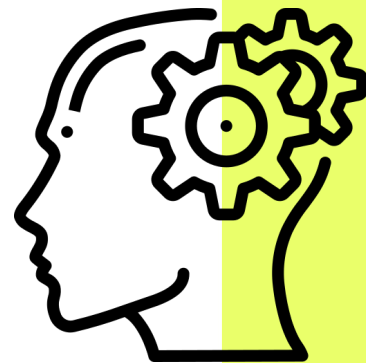
Las listas u otras colecciones son del tipo **compuesto**, por lo que se pasa por referencia, las modificaciones dentro de la función también se harán por fuera.

Fuente: [Hektor Docs](#)

Argumentos valor-referencia

Como vimos, las listas en este caso, hacen referencia a su variable original mientras que los números o tipos de datos más simples “pasan” directamente por valor.

A continuación una pregunta clave...





Para pensar

¿Es posible que de alguna forma le digamos a Python cuándo queremos pasar un argumento por referencia o por valor?

Contesta mediante el chat de Zoom



Argumentos valor – referencia

La respuesta es NO. En Python no se pueden utilizar punteros como en otros lenguajes.

```
>>> def doblar_valor(numero):  
        return numero *= 2  
>>> numero = 10  
>>> numero = doblar_valor(numero)
```

Aunque podemos utilizar trucos, como devolver el valor modificado dentro de la función y volverlo a asignar a la misma variable en caso de desear que sea **"referencia"**.



Break

¡10 minutos y volvemos!

Argumentos indeterminados

Uso de *Args y **Kwargs

¿Para qué se usan?

Lo primero de todo es que en realidad no tienes por qué usar los nombres args o kwargs, ya que se trata de una mera convención entre programadores.

Sin embargo, lo que sí debes usar es el asterisco simple * o doble **.

Es decir, podrías escribir *variable y **variables.

Uso de *Args

Gracias a los *args en Python, podemos definir funciones cuyo número de argumentos es variable. Es decir, podemos definir funciones genéricas que no aceptan un número determinado de parámetros, sino que se “adaptan” al número de argumentos con los que son llamados.



Ejemplo de uso de *Args

```
def suma(*args):  
    s = 0  
    for arg in args:  
        s += arg  
    return s
```

```
suma(1, 3, 4, 2)  
#Salida 10
```

```
suma(1, 1)  
#Salida 2
```

Veamos aquí como *args puede ser iterado, ya que en realidad es una tupla. Por lo tanto iterando la tupla podemos acceder a todos los argumentos de entrada, y en nuestro caso sumarlos y devolverlos.



Ejemplo de uso de *Args

```
def suma(*args):  
    return sum(args)
```

```
suma(5, 5, 3)  
#Salida 13
```

Una forma más sencilla de escribir el código anterior. 🙌

Uso de ****Kwargs**

Al igual que en `*args`, en `**kwargs` el nombre es una mera convención entre los usuarios de Python. Puedes usar cualquier otro nombre siempre y cuando respetes el `**`.

A diferencia de `*args`, los `**kwargs` nos permiten dar un nombre a cada argumento de entrada, pudiendo acceder a ellos dentro de la función a través de un diccionario.



Ejemplo uso de *Kwargs

```
def suma(**kwargs):  
    s = 0  
    for key, value in kwargs.items():  
        print(key, "=", value)  
        s += value  
    return s
```

```
suma(a=3, b=10, c=3)  
#Salida  
#a = 3  
#b = 10  
#c = 3  
#16
```

Podemos ver que es posible iterar los argumentos de entrada con `items()`, y podemos acceder a la clave `key` (o nombre) y el valor o `value` de cada argumento.



Ejemplo uso de *Kwargs

```
def suma(**kwargs):  
    s = 0  
    for key, value in kwargs.items():  
        print(key, "=", value)  
        s += value  
    return s
```

```
suma(a=3, b=10, c=3)  
#Salida  
#a = 3  
#b = 10  
#c = 3  
#16
```

El uso de los `**kwargs` es muy útil si además de querer acceder al valor de las variables dentro de la función, quieres darles un nombre que de una información extra.

Pensar como un diccionario.



Conversiones tipos de datos

Pasaremos de milímetros a metros según el parámetro de la función

Duración: **15 minutos**



ACTIVIDAD EN CLASE

Conversaciones tipos de datos

Descripción de la actividad.

Realiza una función que, dependiendo de los parámetros que reciba, convierta a milímetros o a metros.

1- Si recibe un argumento, supone que son milímetros y convierte a metros, centímetros y milímetros.

2- Si recibe 3 argumentos, supone que son metros, centímetros y milímetros y los convierte a milímetros.

Funciones recursivas

Recursividad

La recursión o recursividad es un proceso de repetición en el que algo se repite a sí mismo. Es el efecto que sucede cuando se ponen dos espejos frente al otro.

En informática, la recursividad es una técnica muy utilizada, la cual se basa en dividir un problema en partes más pequeñas para poder solucionarlo de forma más simple. Donde más se suele utilizar es en las funciones.

Recursividad

Cuando una función se llama a sí misma, tenemos una función recursiva con un comportamiento muy similar al de una sentencia iterativa (if, while, etc.) pero debemos encargarnos de planificar el momento en que dejan de llamarse a sí mismas o tendremos una función recursiva infinita.

Podríamos dividir las funciones recursivas en dos: discursivas sin retorno y discursivas con retorno.



Función recursiva sin retorno

```
>>> def cuenta_regresiva(numero):  
...     numero -= 1  
...     if numero > 0:  
...         print numero  
...         cuenta_regresiva(numero)  
...     else:  
...         print "Boooooooooom!"  
...         print "Fin de la función", numero  
...  
>>> cuenta_regresiva(5)  
4  
3  
2  
1  
Boooooooooom!  
Fin de la función 0  
Fin de la función 1  
Fin de la función 2  
Fin de la función 3  
Fin de la función 4
```

Un ejemplo de una función recursiva sin retorno es el de una cuenta regresiva hasta cero a partir de un número dado.



Función recursiva con retorno

```
>>> def factorial(numero):  
...     print "Valor inicial ->",numero  
...     if numero > 1:  
...         numero = numero * factorial(numero -1)  
...     print "valor final ->",numero  
...     return numero  
...  
>>> print factorial(5)  
Valor inicial -> 5  
Valor inicial -> 4  
Valor inicial -> 3  
Valor inicial -> 2  
Valor inicial -> 1  
valor final -> 1  
valor final -> 2  
valor final -> 6  
valor final -> 24  
valor final -> 120  
120
```

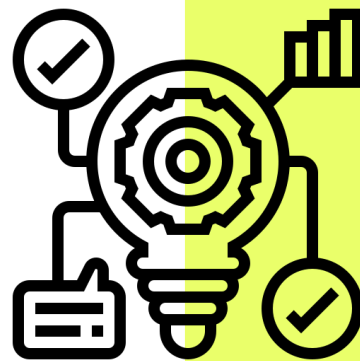
Un ejemplo de una función recursiva con retorno, es el ejemplo del cálculo del factorial de un número corresponde al producto de todos los números desde 1 hasta el propio número.

Funciones integradas

Funciones

Ahora que conocemos las funciones, no podemos acabar sin comentar varias de las integradas en Python. Muchas de ellas son para hacer conversiones entre tipos de datos, otras para manipular información, matemáticas, y de más.

👉 Veremos un resumen de las más utilizadas incluyendo algunas ya conocidas.





Int

La función `int()` devuelve un número entero. Es un constructor, que crea un entero a partir de un entero float, entero complex o una cadena de caracteres que sean coherentes con un número entero.

```
>>> int(2.5)
```

```
2
```

=

```
>>> int("25")
```

```
25
```

Fuente: [EntrenamientoPython](#)



Int

```
>>> int("2.5")
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>
ValueError: invalid literal for int()
with base 10: '2.5'

```
>>> int("doscientos")
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>
ValueError: invalid literal for int()
with base 10: 'doscientos'

La función `int()` solo procesa correctamente cadenas que contengan exclusivamente números.

Si la cadena contiene cualquier otro carácter, la función devuelve una excepción `ValueError`.



Float

```
>>> float(2.5)
2.0
>>> float(25L)
25.0
>>> float("2")
2.0
>>> float("2.5")
2.5
```

La función `float()` devuelve un número coma flotante `float`. Es un constructor, que crea un coma flotante a partir de un entero, entero `long`, entero `float` (cadenas de caracteres formadas por números y hasta un punto) o una cadena de caracteres que sean coherentes con un número entero.



Str

La función `str()` es el constructor del tipo de cadenas de caracteres, se usa para crear un carácter o cadenas de caracteres mediante la misma función `str()`.

Puede convertir un número entero a una cadena de caracteres, de la siguiente forma:

```
>>> str(2.5)
'2.5'
```




Str

Puede convertir un número **float** a una cadena de caracteres, de la siguiente forma:

```
>>> str(2.5)
"2.5"
>>> str(-2.5)
"-2.5"
```

Puede convertir un número **complex** a una cadena de caracteres, de la siguiente forma:

```
>>> str(2.3+0j)
"(2.3+0j)"
```

Puede convertir un tipo **booleano** a una cadena de caracteres, de la siguiente forma:

```
>>> str(True)
"True"
>>> str(False)
"False"
```



Round

La función `round()` redondea un número flotante a una precisión dada en dígitos decimal (por defecto 0 dígitos). Esto **siempre devuelve un número flotante**. La precisión tal vez sea negativa.

👉 En el siguiente ejemplo veremos el redondeo de un número flotante a entero, mayor o igual a .5 al alza:

```
>>> round(2.5)  
3
```

👉 En este otro ejemplo veremos el redondeo de un número flotante a entero, menor de .5 a la baja:

```
>>> round(2.4)  
2
```



Help

Invoca el menú de ayuda del intérprete de Python

```
>>> help()
```

Welcome to Python 3.8! This is the online help utility.

If this is your first time using Python, you should definitely check out the tutorial on the Internet at <https://docs.python.org/3.8/tutorial/>.

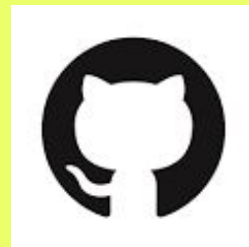
Enter the name of any module, keyword, or topic to get help on writing Python programs and using Python modules. To quit this help utility and return to the interpreter, just type "quit".

To get a list of available modules, keywords, or topics, type "modules", "keywords", or "topics". Each module also comes with a one-line summary of what it does; to list the modules whose summaries contain a given word such as "spam", type "modules spam".
help>

Repositorio Github

Te dejamos el acceso al Repositorio de Github donde encontrarás todo el material complementario y scripts de la clase.

✓ [Repositorio Python](#)





#CoderAlert

Encontrarás en la [Guía de Actividades](#) del curso una actividad para aplicar todo lo aprendido hoy sobre **FUNCIONES** a tu Proyecto. ¡Será fundamental al momento de realizar tu primera pre entrega en clase N°10!

Actividad N° 2



¡Funciones!

Consigna

- ✓ Realizar los ejercicios 1, 2, 3, 4, 5 y 6.

Formato

- ✓ Documento de Word, Google Docs o PDF o mejor aún Colabs.



¡Funciones!

- 1) Realiza una función llamada `area_rectangulo()` que devuelva el área del rectángulo a partir de una base y una altura. Calcula el área de un rectángulo de 15 de base y 10 de altura

👉 Ayuda: El área de un rectángulo se obtiene al multiplicar la base por la altura.

- 2) Realiza una función llamada `area_circulo()` que devuelva el área de un círculo a partir de un radio. Calcula el área de un círculo de 5 de radio


👉 Ayuda: El área de un círculo se obtiene al elevar el radio a dos y multiplicando el resultado por el número pi. Puedes utilizar el valor 3.14159 como pi o importarlo del módulo `math`.



¡Funciones!

- 3) Realiza una función llamada `relacion()` que a partir de dos números cumpla lo siguiente:
- ✓ Si el primer número es mayor que el segundo, debe devolver 1.
 - ✓ Si el primer número es menor que el segundo, debe devolver -1.
 - ✓ Si ambos números son iguales, debe devolver un 0.


Comprueba la relación entre los números: '5 y 10', '10 y 5' y '5 y 5'

- 4) Realiza una función llamada `intermedio()` que, a partir de dos números, devuelva su punto intermedio:
-  **Ayuda:** El número intermedio de dos números corresponde a la suma de los dos números dividida entre 2

Comprueba el punto intermedio entre -12 y 24



¡Funciones!

- 5) Realiza una función llamada `recortar()` que reciba tres parámetros. El primero es el número a recortar, el segundo es el límite inferior y el tercero el límite superior. La función tendrá que cumplir lo siguiente:
- ✓ Devolver el límite inferior si el número es menor que éste
 - ✓ Devolver el límite superior si el número es mayor que éste.
 - ✓ Devolver el número sin cambios si no se supera ningún límite.
 - ✓ Comprueba el resultado de recortar 15 entre los límites 0 y 10
- 4) Realiza una función `separar()` que tome una lista de números enteros y devuelva dos listas ordenadas. La primera con los números pares, y la segunda con los números impares:
-  **Ayuda:** Para ordenar una lista automáticamente puedes usar el método `.sort()`



Primera pre-entrega

En la clase que viene se presentará la consigna de la primera parte del Proyecto final, que **nuclea temas vistos entre las clases 1 y 10**.
Recuerda que tendrás 7 días para subirla en la plataforma.



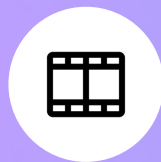
#Codertraining

¡No dejes para mañana lo que puedes practicar hoy! Te invitamos a revisar la [Guía de Ejercicios Complementarios](#), donde encontrarás un ejercicio para poner en práctica lo visto en la clase de hoy.

¿Preguntas?

Resumen de la clase hoy

- ✓ Argumentos y Parámetros
- ✓ Funciones Recursivas
- ✓ Funciones Integradas



¿Quieres saber más?
**Te dejamos material
ampliado de la clase**



MATERIAL AMPLIADO

Recursos multimedia

Título

- ✓ Artículo: [Funciones](#)
- ✓ Artículo: [Funciones Avanzadas](#)
- ✓ Artículo: [Funciones Integradas](#)
- ✓ Artículo: [Funciones Recursivas](#)
- ✓ [RepasoEjercicios](#)

Opina y valora
esta clase

Muchas gracias.

#DemocratizandoLaEducación