TEL101 - Iniciación a la Programación

Clase 6: Módulos y Funciones

Prof. Nicolás Torres
nicolas.torresr@usm.cl
Ingeniería Civil Telemática
Departamento de Electrónica
Universidad Técnica Federico Santa María

Funciones

Bloque de código que cumple una tarea específica.

En general, recibe al menos un valor como entrada, conocido como **parámetro**, y entrega un valor como salida, llamado **retorno**. Cada vez que se utiliza una función, se realiza un "llamado" al bloque de código que la compone a través del nombre y se le entrega los **argumentos**, entre paréntesis y separados por comas, que se almacenan en los parámetros definidos.

Funciones Integradas

Python incluye un conjunto de funciones integradas con el lenguaje que siempre disponibles y pueden ser utilizadas en cualquier momento.

- Entrada y Salida de datos: `input()` y `print()`
- Transformadoras de datos: `int()`, `float()`, `list()`, `str()`, etc.
- Funciones matemáticas simples: `abs()`, `pow()`, `sum()`, `min()`, `max()`, etc.

In [1]:
```
abs(4-5)
```

Out[1]:
```
1
```

In [2]:
```
pow(2,3)
```

Out[2]:
```
8
```

In [3]:
```
max(3, 2, -4, 4, 1, -5, 0, 8, 2, -1)
```

Out[3]:
```
8
```

In [4]:
```
round(5.76543, 2)
```

Out[4]:
```
5.77
```

Módulos

- Existen muchas otras funciones implementadas en Python.
- Sin embargo, no están disponibles de forma inmediata para su uso.
- Se encuentran en archivos aparte, conocidos como **bibliotecas**.
- Para acceder a estas funciones, es necesario incluir su código con la palabra reservada `import`.

Biblioteca `math`

Permite acceder a funciones matemáticas como la raíz cuadrada, la constante π, logaritmo, etc.

In [5]:
```
import math

print("La raiz cuadrada de 16 es",math.sqrt(16))
print("π =",math.pi)
print("sen(90) =",math.sin(math.pi/2))
print("cos(90) =",math.cos(math.pi/2))
print("log(8,2)=",math.log(8,2))
```

```
La raiz cuadrada de 16 es 4.0
π = 3.141592653589793
sen(90) = 1.0
cos(90) = 6.123233995736766e-17
log(8,2)= 3.0
```

Importar todo desde...

Para importar todas las funciones desde un módulo directamente al programa se utiliza la palabra reservada `from`, el nombre del módulo y el asterisco `*` para incluir todas las funciones del módulo. Por lo tanto, ahora se pueden llamar como si estuvieran definidas en el mismo programa.

In [6]:
```
from math import *

print("La raiz cuadrada de 16 es",sqrt(16))
print("π =",pi)
print("sen(90) =",sin(pi/2))
print("cos(90) =",cos(pi/2))
print("log(8,2)=",log(8,2))
```

```
La raiz cuadrada de 16 es 4.0
π = 3.141592653589793
sen(90) = 1.0
cos(90) = 6.123233995736766e-17
log(8,2)= 3.0
```

Al importar de esta forma, debe tener cuidado con la coincidencia de nombres de variables y funciones en su código.

Alias

También es posible agregar bibliotecas y/o sus funciones con un **alias**.

In [7]:
```
import math as m
print("La raiz cuadrada de 16 es",m.sqrt(16))
```

```
La raiz cuadrada de 16 es 4.0
```

In [8]:
```
from math import sqrt as raiz
```

```
print("La raiz cuadrada de 16 es",raiz(16))
```

```
La raiz cuadrada de 16 es 4.0
```

Biblioteca `random`

Se utiliza para generar números aleatorios (al azar).

In [9]:
```
from random import randint, choice, shuffle
```

In [10]:
```
print(randint(1,10))
print(randint(1,10))
print(randint(1,10))
```

```
9
3
2
```

In [11]:
```
print(choice(['a','b','c','d','e','f']))
print(choice(['a','b','c','d','e','f']))
print(choice(['a','b','c','d','e','f']))
```

```
f
e
c
```

In [12]:
```
lista = [1,2,3,4,5]
shuffle(lista)
print(lista)
```

```
[4, 1, 5, 2, 3]
```

Otras bibliotecas de interés para el futuro

- `TensorFlow` : Para aprendizaje automático.
- `Keras` : Redes Neuronales.
- `matplotlib` : Para gráficas en general (https://matplotlib.org).
- `numpy` : Para funciones matemáticas avanzadas (https://www.numpy.org).
- `seaborn` : Para visualización de datos (basada en matplotlib).
- `scipy` : Estadística y y algoritmos matemáticos.
- `sklearn` : Para aprendizaje automático.
- `pandas` : Para manipulación y análisis de datos.
- `os` : Para interactuar con el Sistema Operativo.

Además, un usuario puede crear sus propios módulos.

Creación de módulos

En un archivo auxiliar, ubicado en la misma ruta donde tiene su `main.py` , puede tener definidas sus funciones y luego importarlas dentro del archivo principal. Esto permite modularizar el código y facilita la depuración (identificación y corrección de errores).

Creación de funciones

Además de las funciones integradas, es posible **definir** funciones propias y "llamarlas" en cualquier momento.

Definición

Una función se define con la palabra reservada `def`, seguida de un nombre, y los parámetros separados por comas se ubican dentro de los paréntesis. Como todo bloque de código, al final de la línea se requiere el uso de los dos puntos, y todas las instrucciones que realiza la función para entregar un resultado final como retorno deben estar indentadas en el mismo nivel.

```
def nombre_funcion(parámetro1, parámetro2, parámetro3,...):
        # Instrucciones
        return valor
```

Ejemplo

In [13]:
```
def sumar(a, b, c):
    suma = a + b + c
    return suma
```

Llamado

Una vez definida, una función puede ser utilizada como cualquier otra función integrada, todas las veces que se requiera.

Al llamar a una función la ejecución del programa se traslada al bloque de código que la define, y los parámetros reciben los valores de entrada (argumentos). Cuando la función termina, la ejecución vuelve al punto del llamado, llevando consigo el valor de retorno.

In [14]:
```
print("La suma de 1 + 2 + 3 =",sumar(1,2,3))
```

```
La suma de 1 + 2 + 3 = 6
```

In [15]:
```
n1 = int(input("Ingrese un número: "))
n2 = int(input("Ingrese un número: "))
n3 = int(input("Ingrese un número: "))
resultado = sumar(n1,n2,n3)

print("La suma de",n1,"más",n2,"más",n3,"es",resultado)
```

```
Ingrese un número: 1
Ingrese un número: 2
Ingrese un número: 3
La suma de 1 más 2 más 3 es 6
```

In [16]:
```
print(sumar(2.5,4.25,0.25))
print(sumar('1','2','3'))
print(sumar([1,2,3],[4,5],[6,7,8,9]))
```

```
7.0
123
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Visualizar la ejecución

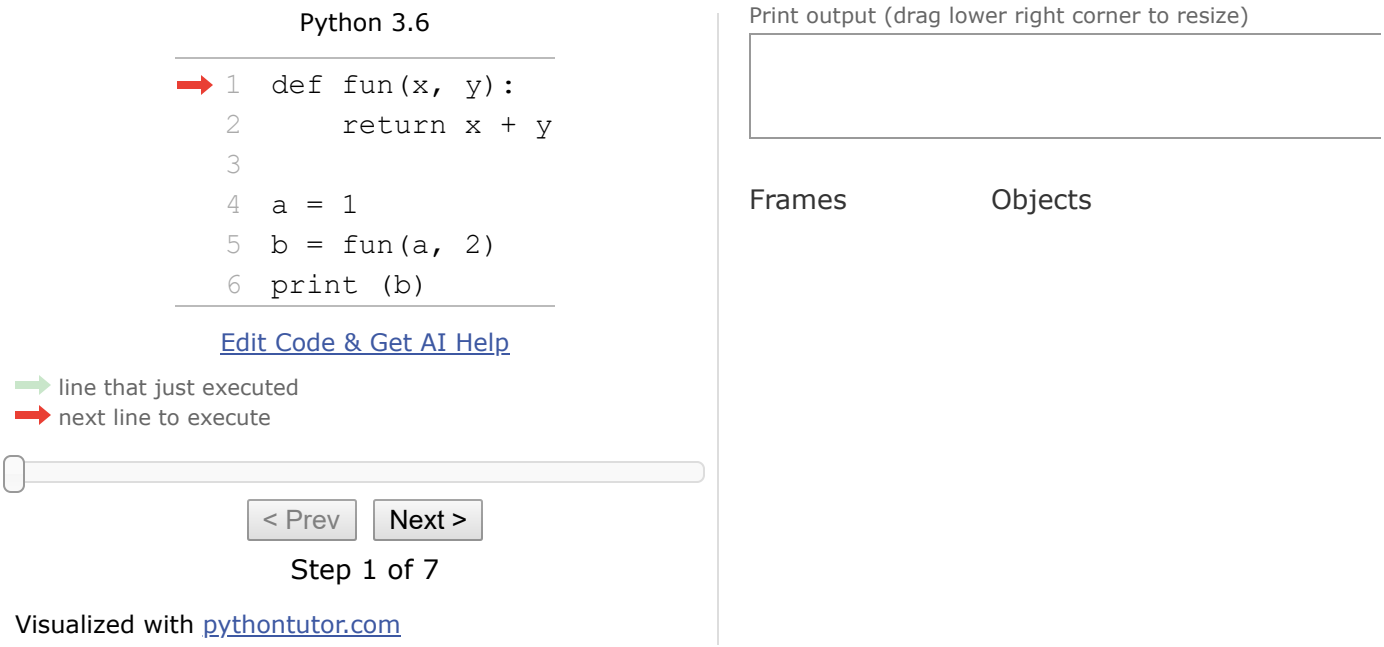
Cuando se trabaja con funciones, la ejecución de un programa puede ser difícil de leer porque ocurren saltos entre una línea y otra (se puede pasar desde la última línea a la primera y luego devolverse).

In [17]:
```
import IPython

src = 'http://pythontutor.com/iframe-embed.html#code=def%20fun%28x,%20y%29%3A%0A%20%20%20%20retu

IPython.display.IFrame(src, width=974, height=550)
```

Out[17]:



Variables locales y globales

Al definir nuestras propias funciones debemos saber identificar el **ámbito** de las variables. El ámbito es "donde vive" la variable. De esta forma, se definen dos tipos de variables y sus consecuentes ámbitos:

- **Variable local:** son aquellas variables creadas dentro de una función (incluye a los parámetros) y tienen un ámbito local, vale decir, solo pueden ser usadas y modificadas dentro de la función.
- **Variable global:** son aquellas variables definidas fuera de cualquier función. Tienen un ambito global, es decir, pueden ser utilizadas en cualquier parte del programa. Sin embargo, no pueden ser modificadas dentro de alguna función (en ese caso se crearía una variable local con el mismo nombre).

Variable global: El valor de π se trata como variable **global**.

In [18]:
```
PI_global = 3.14159265359

def area(radio):
    return PI_global * radio ** 2
```

In [19]:
```
print("El área de un círculo de radio 2 es:",area(2))
```

```
El área de un círculo de radio 2 es: 12.56637061436
```

In [20]:
```
print("El valor de π es:",PI_global)
```

```
El valor de π es: 3.14159265359
```

Variable local: El valor de π se trata como variable **local**.

In [21]:
```
def area(radio):
    PI_local = 3.14159265359
    return PI_local * radio ** 2
```

In [22]:
```
print("El área de un círculo de radio 2 es:",area(2))
```

```
El área de un círculo de radio 2 es: 12.56637061436
```

In [23]:
```
print("El valor de π es:",PI_local)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [23], in <cell line: 1>()
----> 1 print("El valor de π es:",PI_local)

NameError: name 'PI_local' is not defined
```

Diferencias entre `print()` y `return`

La función `print()` y la instrucción `return` pueden tener un comportamiento muy parecido, pero son **completamente** diferentes.

- `return` es el **resultado de una función**, por lo tanto solo debe ir dentro de una función.
- `print()` es una **salida de un programa**, puede ir tanto en una función (aunque no es recomendable) como en el código de un programa principal (el main).

Ejemplo

In [24]:
```
def suma(x,y):
    return x+y
```

Al ejecutar la función no nos mostrará ningun resultado por pantalla, ya que solo estamos creando el "*molde*" de la función. **No la estamos usando aún**.

Para mostrar el resultado de la función por pantalla, se debe llamar a la función y utilizar un `print()` para imprimir su resultado:

In [25]:
```
print(suma(1,2))
```

```
3
```

Sin embargo, si en lugar de utilizar `return` se utiliza un `print()` dentro de la función:

In [26]:
```
def suma(x,y):
    print(x+y)

print(suma(1,2))
```

```
3
None
```

La función retorna el tipo de dato `None` que significa literalmente "Nada", y ya no será posible guardar el resultado de la función porque se imprimirá y se perderá (la función pierde su propósito).

Por lo tanto, como regla general:

- No utilizaremos `input` dentro de una función. Las entradas que ingresan las personas que usan el programa son leídas en el código principal, fuera de las funciones.
- No utilizaremos `print` dentro de una función. Las funciones entregan sus resultados a través de `return`, y en el código principal los resultados son mostrados a través de `print` de ser necesario.

La otra clara diferencia entre `print()` y `return` tiene que ver con la ejecución del código. El `return` termina automáticamente la ejecución del bloque de código dentro de la función.

Entonces, por ejemplo, es posible encontrarse con la siguiente función:

In [27]:
```
def paridad(numero):
    if numero % 2 == 0:
        return "par"
    return "impar"
```

In [28]:
```
print(paridad(1))
print(paridad(2))
print(paridad(3))
print(paridad(4))
```

```
impar
par
impar
par
```