Sesión 05: Funciones en Python

Contenidos:

- Funciones
- Paso de argumentos
- Module
- Package
- Funciones Lambda

FUNCIONES

```
def suma(a,b):
    return(a+b)
def sumaresta(a,b):
    return (a+b),(a-b)
def mostrar():
    print(«suma: » + str(suma(10,5)))
```

Python: Funciones

- **DEFINICIÓN DE FUNCIONES**
  - Se define con la instrucción **def** seguido de un ***nombre de función descriptivo*** seguido de paréntesis de apertura y cierre**()** seguido de parámetros (si fuese necesario) y finaliza con dos puntos **:**

```
#func. con dos parametros
def media (a,b):
    media = (a + b)/2
    return(media)
```

```
#func. con 2 parametros con
#un valor por defecto (b=10)
def suma (a,b=10):
    s=(a + b)
    print("suma" + str(s))
```

```
#function sin parametros
def saluda ():
    print("Hola Mundo")
```

```
#function con varios param.
def suma (*vocales):
    s=len(vocales)
    print("s=" + str(s))
```

Funciones

Encapsulación de código para reutilizar

- Porciones de código que se encapsulan para ser ejecutadas posteriormente (reutilización de código).
- Existen numerosas funciones ya hechas y el usuario puede hacer propias.
- La salida puede ser:
  a) Un valor u objeto
  b) Sin salida, pero ha realizado operaciones.
- Pueden tener argumentos, que son utilizados para realizar los cálculos de las operaciones.

$$f(x) = x^2$$

```
def my_function(param1, param2, ...):
    pass
```

Funciones

- Palabra reservada **def** + nombre de la función + paréntesis ()
- **Argumentos entre paréntesis**:
  a) Son opcionales
  b) Si alguno tiene valor por defecto se indica con '='
  c) Los argumentos con valor por defecto deben figurar siempre después de los requeridos, **nunca antes**.
- Se devuelve el resultado de la función con **return**.

```
def nombre_funcion(arg1, arg2 = True, ...):
        <instrucciones>
        return output
```

Funciones

- Las funciones se pueden crear en cualquier momento del programa
- El código de una función **se ejecuta en un scope propio**.
  - Se tendrá acceso a las variables declaradas dentro del scope de la función.
  - Las funciones no tienen acceso a variables de otras funciones.

```
edad = 19

def mayoria_edad(edad):
        if edad >= 18:
            return True
        else:
            return False

mayoria_edad(edad)
>> True
```

Funciones

- Todas las funciones devuelven un valor:
  - Si no hay return, devuelve el valor especial None.
  - No es necesario declarar los tipos de los valores a devolver.
- No existe la sobrecarga de funciones.
  - No puede haber dos funciones con el mismo nombre aunque tenga diferentes argumentos.
- Las funciones se pueden usar como cualquier tipo de datos. Pueden usarse como:
  - Argumento para otras funciones.
  - Valores devueltos por otras funciones.
  - Asignaciones a variables.
  - Elementos en tuplas, listas…

Funciones

- Devolución de valores

```
# Crearemos ahora una función que devuelve un valor
# Ejemplo sencillo: si es par muliplicamos por 2
# Si es impar, sumamos dos
def transforma_numero(numero):
    if (numero % 2 == 0):
        return(numero * 2)
    else:
        return(numero + 2)
# Aquí acaba La fúnción

#Pruebas
prueba1 = transforma_numero(15)
prueba2 = transforma_numero(12)
print(str(prueba1) + "---" + str(prueba2))
```

```
17---24
```

Funciones integradas

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

https://docs.python.org/3/library/functions.html

Python: Funciones

- **INVOCACION DE FUNCIONES**
  - Una función no es ejecutada hasta que sea invocada.
  - Para invocar a la función simplemente **se llama por su nombre**.

```
def main ():
    #llamada a funciones
     m= media(10,15)
     s= suma(10,15)
    print("suma" + str(s))
    print("media" + str(m))
```

```
def media (a,b):
    media = (a + b)/2
    return(media)
```

```
def suma (a,b):
    s=(a + b)
    print("suma" + str(s))
```

Ejemplo 1 – uso de funciones

Escriba un programa que permita sumar los primeros n números positivos.

- Datos de entrada: Ninguno
- Datos de salida: Impresión de los cinco primeros números positivos
- Utilice funciones

```
def sumador(n): #function con un parametro
  suma = 0
  contador = 0
  while (contador < n):
    contador = contador + 1
    suma = suma + Contador
  return(suma)
#
def main(): #function sin parametros
  n = int(input("Ingrese un numero: "))
  suma = sumador(n) #invoca a la function sumador
  print("La suma de los " + str(n) + " numeros es " + str(suma))
#
main() #invoca a la function main
```

Ejemplo 2 – uso de funciones

Escriba un programa que permita escribir los primeros n números positivos.

- Datos de entrada: Ninguno
- Datos de salida: Impresión de los cinco primeros números positivos
- Utilice funciones

```
def mostrarPalabras(*frase):
  longitud = len(frase)
  for palabra in frase:
    print(palabra)
  return(longitud)

#
def main():
  l = mostrarPalabras("hola", "cómo", "estás")
  print("Numero Palabras = " + str(l))
#
main()
```

PASO DE ARGUMENTOS

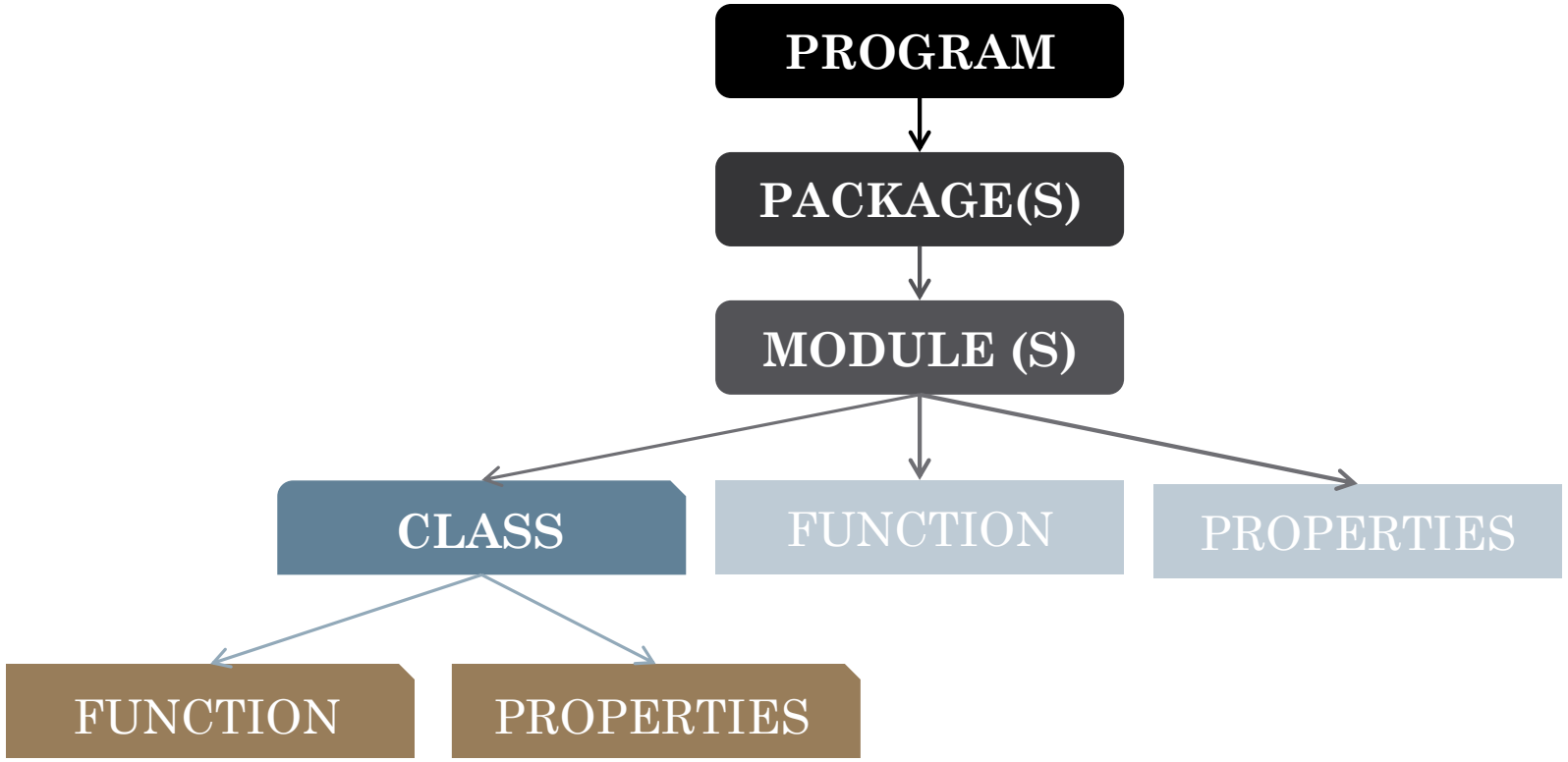
```
def cambiar(l):
    l.append([1,2,3,4])
def main():
    l = [10,20,30]
    cambiar(l)
    print(l)
```

```
def main():
    l = [10,20,30]
cambiar(l)
    print(l)
#
main()
```

Paso de Argumentos

- Los módulos en Python son simplemente archivos con la extensión ***".py"***, en el cual se implementa un conjunto de funciones definidas por el usuario.
- Los módulos son importados desde otros módulos utilizando la instrucción ***import***.
- La primera vez que el módulo es cargado en un script de ejecución Python, es inicializado para ser ejecutado en el modulo una vez.
  - Si otro modulo importa el mismo modulo, no será cargado otra vez, pues ya se encuentra en memoria.
  - Las variables locales dentro del modulo funcionan como un Singleton.

Organización de una solución Python



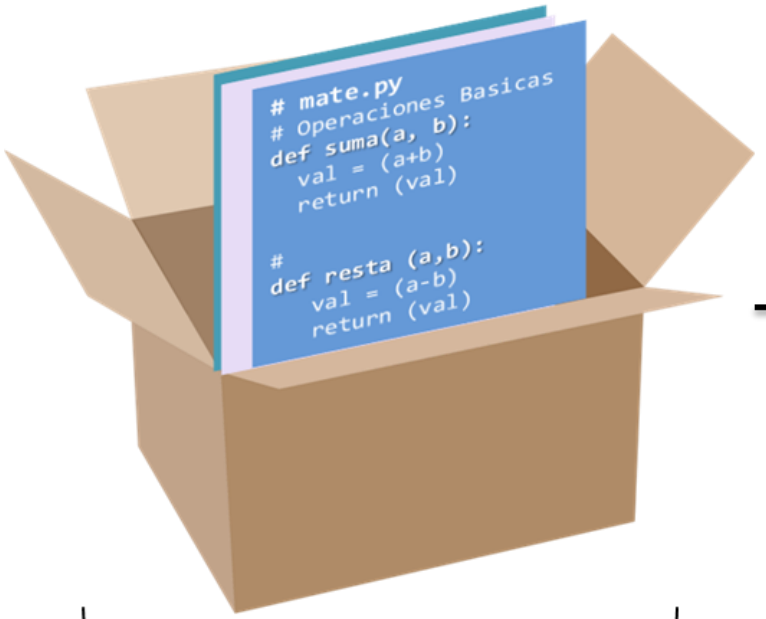
MODULES (MODULOS)

```
from math import *

import math as m

from math import pow
```

Module/Modulo

- Los módulos en Python son simplemente archivos con la extensión ***".py"***, en el cual se implementa un conjunto de funciones definidas por el usuario.
- Los módulos son importados desde otros módulos utilizando la instrucción ***import***.
- La primera vez que el módulo es cargado en un script de ejecución Python, es inicializado para ser ejecutado en el modulo una vez.
  - Si otro modulo importa el mismo modulo, no será cargado otra vez, pues ya se encuentra en memoria.
  - Las variables locales dentro del modulo funcionan como un Singleton.

Module/Modulo



Importando la función suma del modulo "mate.py"

```
from mate import suma

def opera(a, b):
  s = suma(a,b)
  print(str(s))
#
opera(3,5)
```

Paquete / Módulos(archivos)

Módulo importando un paquete

MODULE (MODULOS): SINTAXIS

- Bucle con condición controlada
- La ejecución implica dos aspectos:
  - **EVALUACIÓN DE LA CONDICIÓN**
    - **La condición es:**
      - **Una EXPRESION BOOLEANA.**
      - *Tiene dos posibles resultados excluyentes verdadero o falso)*
  - **EJECUCION DEL CICLO**
    - Si la condición resulta ser verdadera se lleva a cabo la **EJECUCIÓN DE LAS INSTRUCCIONES** del cuerpo del bucle.

MODULES(MODULOS): IMPORTACION

mate.py

```
# mate.py
# Operaciones Basicas
#
def suma(a, b):
  return (a+b)

#
def resta (a,b):
 return (a-b)

#
def multiplicacion(a, b):
  return (a*b)

#
def resta (a,b):
 if(b != 0 )
   return (a/b)
 else
   return(0)
```

Formas de la instrucción: **IMPORT**

```
import mate

def opera(a, b):
   s = mate.suma(a,b)
  print(str(s))
#
opera(3,5)
```

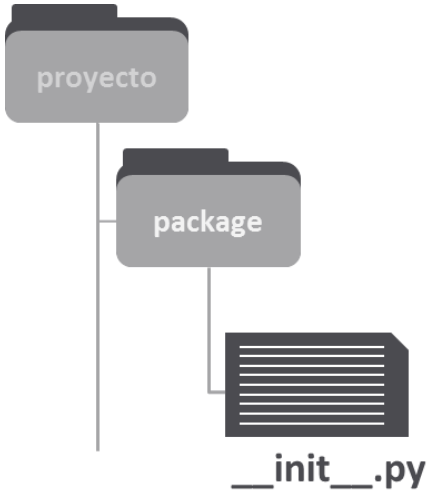
```
import mate as m

def opera(a, b):
   s = m.suma(a,b)
  print(str(s))
#
opera(3,5)
```

Formas de la instrucción: **FROM .. IMPORT**

```
from mate import *

def opera(a, b):
   s = suma(a,b)
  print(str(s))
#
opera(3,5)
```

```
from mate import suma

def opera(a, b):
   s = suma(a,b)
  print(str(s))
#
opera(3,5)
```

PACKAGE (PAQUETES)




```
__init__.py

from pkg import module.function
```

PACKAGE(PAQUETE)

- Un paquete es una **estructura de carpetas**.
  - Es una manera de organizar un conjunto de módulos relacionados dentro de una estructura jerárquica similar a un árbol.
  - Mantiene el código organizado evitando colisiones de nombres
- **Define un entorno** o ambiente para una **aplicación python única** la cual **consiste de módulos, subpaquetes y subsubpaquetes** y así sucesivamente.

Funciones Lambda

Funciones Lambda

Creando expresiones en llamadas a funciones.

Las funciones lambda:

- ✓ Son funciones de **una sola expresión**
- ✓ Usan la palabra reservada **lambda**
- ✓ No necesitan estar ligadas a un nombre
- ✓ Se pueden definir en el lugar donde se usan
- ✓ Poseen un **return** implícito

Funciones lambda

Creando expresiones en llamadas a funciones

Las funciones lambda :

- Son funciones de **una sola expresión**
- Usan la palabra reservada **lambda**
- No necesitan estar ligadas a un nombre
- Se pueden definir en el lugar donde se usan
- Poseen un **return** implícito
- Sintaxis:

lambda x, y: x + y

Funciones lambda

```
una_funcion_lambda = lambda arg: <operación con arg>

una_funcion_lambda(arg = <valor>)
>> ---resultado de operación con<valor>---



[x ** 2 for x in [1, 2, 3]]
>> [1, 4, 9]

list(map(lambda x: x ** 2, [1, 2, 3]))
>> [1, 4, 9]
```

Ejemplos de funciones lambda

```
>>> doblar = lambda numero: numero*2
>>> doblar(2)
4
```

```
>>> revertir = lambda cadena: cadena[::-1]
>>> revertir("Plone")
'enolP'
>>> revertir("enolP")
'Plone'
```

```
>>> impar = lambda numero: numero%2 != 0
>>> impar(5)
True
```

```
>>> sumar = lambda x,y: x+y
>>> sumar(5,2)
7
```

Funciones Lambda

```
def IGV(monto):
        impuesto = monto*0.18
        return impuesto

IGV(200)
>> 36
```

```
IGV2 = lambda x: x*0.18

IGV2(200)
>> 36
```

```
(lambda x: x*0.18)(200)
>> 36
```


go raibh maith agat
мерси
matondo asante
köszönöm
mersi
dhanyavad
rahmet
hvala
najis tuke
mochchakkeram
barka
ngiyabonga
kia ora
mamnun
gracias
sobodi
tanemirt
bayarlalaa
vinaka
dziękuję
nandri
sagolun
chokrane
rahmat
dank je
aciu
grazie
gratias ago
dankie
enkosi
shukriya
gràcies
paldies
sukriya
kam sah hamnida
chnorakaloutioun
blagodaram
terima kasih
taiku
teşekkür ederim
mauruuru
didi madloba
faafetai lava
kop khun krap
obrigada
djiere dieuf
arigatô
thank you
akun
mahalo
murakoze
mèsi
spas
graçie
dakujem
tau
спасибо
trugarez
dankon
tenki
obrigado
bedankt
diolch
grazzi
takk
sulpáy
merci
ευχαριστώ
dhanyavadagalu
xièxie
dêkuji
mercé
tack
manana
welalin
misaotra
nanni
ありがとう
tapadh leat
kiitos
謝謝
danke