Haz una copia antes de editarla

GOBIERNO DE EL SALVADOR

SECRETARÍA DE INNOVACIÓN

Agenda
Sesión 9/18

1. **Punto 1:** funciones
2. **Punto 2:** valores de entrada y salida
3. **Punto 3:** args keyword
4. **Punto 4:** alcance global
5. **Punto 5:** expresiones lambdas
6. **Punto 6:** función map
7. **Punto 7:** funciones anidadas

PROGRAMACIÓN CON PYTHON

FUNCIONES

Funciones - introducción

Son el primer paso para comprender la POO con Python.

Contienen el código necesario para realizar una tarea específica.

Podemos reutilizarla x cantidad de veces.

Definición (creación) y llamada

Funciones – definición (creación)

```
#definiendo (creando una función)
def bienvenida():
    print('Bienvenidos a Python')

#llamando la función
bienvenida()
```

Funciones – valores de entrada y salida

Las funciones toman datos de entrada y retornan resultados.

Los datos de entrada son conocidos como parámetros.

Funciones – valores de entrada y salida

```
def nombre_de_la_funcion(variable): #función que necesita datos de entrada (parámetros)
    variable2 = “Mensaje {} complemento mensaje”.format(variable)
    return variable2

variable3 = nombre_de_la_funcion(variable=argumento)# parámetro tendrá como valor el argumentos
print(variable3)
```

Funciones – valores de entrada y salida

```
def nombre_de_la_funcion(variable):
    mensaje = 'Bienvenidos {} al curso de Python'.format(variable)
    return mensaje

llamada_funcion=nombre_de_la_funcion('todos los grupos')
print(llamada_funcion)
```

Funciones – valores de entrada y salida

Reduciendo código

```
def nombre_de_la_funcion(variable):
 return 'Bienvenidos {} al curso de Python'.format(variable)


print(nombre_de_la_funcion('todos los grupos'))
```

Funciones – valores de entrada y salida

```
def nombre_de_la_funcion(variable):
    mensaje = 'Bienvenidos {} al curso de Python'.format(variable)
    #mensaje = f'Bienvenidos {variable} al curso de Python'
    print(mensaje) #en lugar de return

nombre_de_la_funcion('todos los grupos')
```

Funciones – valores de entrada y salida

```
def promedio(num1, num2, num3): #N cantidad de parámetros
        return (num1+num2+num3)/3

elpromedio = promedio(7,8,10)
print(elpromedio)
```

Funciones – valores de entrada y salida

```
def multiple():
    return "Un string", "otro string", 10, 10.5, True #N cantidad de valores

val1, val2, val3, val4, val5 = multiple()
print(val1, val2, val3, val4, val5)
```

Funciones – valores de entrada y salida

```
def registro(nombre, apellido, edad, sexo):
        return { #objeto diccionario
            'nombre' : nombre,
            'apellido' : apellido,
            'edad' : edad,
            'sexo' : sexo  # sin coma
        }

llamada = registro()

print(llamada["nombre"])
print(llamada["apellido"])
print(llamada["edad"])
print(llamada["sexo"])
```

Funciones – valores de entrada y salida

```
def registro(nombre, apellido, edad, sexo): #parámetros
        return { #objeto diccionario
            'nombre' : nombre,
            'apellido' : apellido,
            'edad' : edad,
            'sexo' : sexo  # sin coma
        }

llamada = registro("Francisco", "Quezada", 33, "Hombre") #argumentos

print(llamada["nombre"])
print(llamada["apellido"])
print(llamada["edad"])
print(llamada["sexo"])
```

Funciones – valores de entrada y salida

Python nos permite asignar valores por defecto a los parámetros. Pocos lenguajes permiten hacer esto.

```
def registro(nombre, apellido, edad, sexo="Hombre"):
        return {
            'nombre' : nombre,
            'apellido' : apellido,
            'edad' : edad,
            'sexo' : sexo
        }

llamada = registro("Francisco", "Quezada", 33)
print(llamada["nombre"])
print(llamada["apellido"])
print(llamada["edad"])
print(llamada["sexo"])
```

Funciones – valores de entrada y salida

Si asignamos todos los parámetros por default ya no necesitaremos colocar argumentos.

```
def registro(nombre='', apellido='', edad='', sexo="Hombre"):
        return {
            'nombre' : nombre,
            'apellido' : apellido,
            'edad' : edad,
            'sexo' : sexo
        }

llamada = registro()
print(llamada["nombre"])
print(llamada["apellido"])
print(llamada["edad"])
print(llamada["sexo"])
```

Funciones – valores de entrada y salida

Podemos asignar argumentos directamente sin respetar el orden de los valores (parámetros).

```
def registro(nombre='', apellido='', edad='', sexo="Hombre"):
        return {
            'nombre' : nombre,
            'apellido' : apellido,
            'edad' : edad,
            'sexo' : sexo
        }

llamada = registro(edad=33,apellido="Quezada",nombre="Francisco")
print(llamada["nombre"])
print(llamada["apellido"])
print(llamada["edad"])
print(llamada["sexo"])
```

Funciones – valores de entrada y salida

```
nombre_digitado=input("Digita tu nombre: ")

def function_nombre(curso, nombre, inicio):
        print("Hola " + nombre + " bienvenido al curso de " + curso)

function_nombre(inicio="31/11/2020",curso="Python 3",nombre=nombre_digitado)
```

PROGRAMACIÓN CON PYTHON

ARGS KEYWORD

Args keyword

O Keyword Arguments (argumentos de palabras clave) y en algunos textos kwargs

Habrá ocasiones en las que no sabemos el número de parámetros que recibirá una función.

Args keyword - sintaxis

Habrá ocasiones en las que no sabemos el número de parámetros que recibirá una función.

Colocamos un solo parámetro anteponiendo el signo *

Por convención entre la comunidad Python se utiliza **args**, pero podemos utilizar cualquier palabra

```
def suma(*args):
        sumatotal = 0
        for valoresasumar in args:
                sumatotal+=valoresasumar
        print(sumatotal)

sumatotal=suma(10,15) #podremos agregar más

################

def suma(*args):
        sumatotal = 0
        for valoresasumar in args:
                sumatotal+=valoresasumar
        return sumatotal

resultado = suma(10,15)
print(resultado)
```

Args keyword - sintaxis

El * agrupa todos los argumentos dentro de una tupla y los valores de la tupla son asignados al parámetro.

```
def suma(*args):
        sumatotal = 0
        print(args)

        for valoresasumar in args:
                sumatotal+=valoresasumar
        print(sumatotal)

sumatotal=suma(10,15,35.5)
```

Args keyword

Aunque utilicemos *args no nos limita a que podamos hacer uso de más parámetros.

```
def suma(otro_parametro,*args):
        sumatotal = 0
        print(otro_parametro)
        print(args)

        for valoresasumar in args:
                sumatotal+=valoresasumar
        print(sumatotal)

sumatotal=suma("Este es el valor del otro parámetro",10,15,35.5)
```

Args keyword

**kwargs

Nos agrupa los resultados en diccionarios {}

```
def nombrecompleto(**kwargs):
        print(kwargs)
nombrecompleto(nombre="Francisco", apellido="Quezada")
```

Args keyword

****kwargs**

Si se desconoce el número de argumentos de palabras clave, agregue un doble ** antes del nombre del parámetro:

Args keyword

```
def nombrecompleto(**kwargs):
        print("Mi nombre completo es " + kwargs["nombre"]+" "+kwargs["apellido"])

nombrecompleto(apellido="Quezada", nombre="Francisco", edad=33, sexo="Hombre")

############

def nombrecompleto(**kwargs):
        print(kwargs)
nombrecompleto(nombre="Francisco", apellido="Quezada")
```

Args keyword

```
def impresionmultiple(obligatorio,*args,**kwargs):
        print(obligatorio)
        print(args)
        print(kwargs)

impresionmultiple("Valor obligatorio",5,4,"lunes",10.5,nombre="Francisco",edad=33)
```

PROGRAMACIÓN CON PYTHON

ALCANCE GLOBAL

Alcance global

Consiste en el comportamiento de las variables dentro y fuera de una función.

```
color = 'Azul' #variable declarada fuera de la función

def muestra_color():
        print(color)

muestra_color()
print(color)

#a las variables declaradas fuera de una función se les conoce como variables globales.
```

Alcance global

```
color = 'Azul'

def muestra_color():
        color = 'Verde' #variable declarada dentro de la función
        print(color)

muestra_color()
print(color)

#a las variables declaradas dentro de una función se les conoce como variables locales
```

Alcance global

Los namespace (espacio de nombres) pueden coexistir en Python.

Para Python las 2 variables son totalmente diferentes si tienen el mismo nombre fuera y dentro de una función.

Se puede tener acceso a la variable global en todo el script hasta que el script termine, en cambio a la variable local solo se puede tener acceso solo dentro de la función hasta que ésta termine.

Alcance global

La palabra reservada global nos permitirá modificar una variable global dentro de una función.

```
color = 'Azul'

def muestra_color():
        global color
        color = 'Amarillo'
        print(color)

muestra_color()
print(color)
```

PROGRAMACIÓN CON PYTHON

EXPRESIONES LAMBDA

Expresiones lambda

Lambda o “pequeña función anónima” puede tomar cualquier número de argumentos. Generalmen

```
def calcula_area_triangulo(base,altura):
        return base*altura/2

variable = calcula_area_triangulo #asignamos la función a la variable
area = variable(3,5)
print(area)
print(variable(3,5))
print(calcula_area_triangulo(3,5))
```

Expresiones lambda

Lambda o “pequeña función anónima” puede tomar cualquier número de argumentos. Generalmente una función lambda realiza tareas básicas o pequeñas.

```
funcion_lambda = lambda base,altura : base * altura / 2

#llamando a la función
area = funcion_lambda(3,5)
print(area)
print(funcion_lambda(3,5))

#Las llamamos “funciones anónimas” porque
técnicamente carecen de nombre (función sin nombre).
#Para poder utilizarlas debemos auxiliarnos de una variable
```

PROGRAMACIÓN CON PYTHON

FUNCIÓN MAP

Función map

La función map() ejecuta una función específica para cada elemento en un iterable. Los items se envían a la función como parámetro.

```
def cuadrado(numero):
 return numero * numero

lista = [1,2,3,4,5]

#map(función, iterables)
resultado = map(cuadrado, lista)

lista_resultado = list(resultado)
print(lista_resultado)
```

Función map

```
def recorre_string(a):
  return len(a)

x = map(recorre_string, ('manzana', 'melón', 'pera'))

print(x)

#convirtiendo el mapa en una lista:
print(list(x))
```

PROGRAMACIÓN CON PYTHON

FUNCIONES ANIDADAS

Funciones anidadas

Python permite anidar funciones (funciones dentro de otra función) igual que los ciclos y estructuras condicionales.

```
def lista_alumnos(listado):
    def mostrar_lista():
        for alumno in listado:
            print(alumno)
    mostrar_lista()
    print(listado) #imprime el parámetro de la función lista_alumnos

listado = ['Luis', 'Ana', 'Carlos', 'Laura', 'Rafael', 'Frank']
lista_alumnos(listado)
```

RESUMEN DE SESIÓN

GOBIERNO DE
EL SALVADOR

SECRETARÍA DE
INNOVACIÓN