Práctica 1

MIS PRIMERAS FUNCIONES

Debe terminarla para el domingo 8 de diciembre

Esta práctica consta de dos partes:

- Primera parte: Repaso de conceptos de funciones.

El objetivo de la primera parte es que usted repase los conceptos estudiados acerca del desarrollo de funciones. Siguiendo las instrucciones usted escribirá y estudiará partes de códigos de programas fuentes para ir reforzando el desarrollo de funciones. Los conceptos aprendidos son fundamentales para el aprendizaje de programación. Debe responder y analizar respuestas.

- Segunda parte: Desarrollo de funciones.

Continúe con la segunda parte hasta que domine el material de la primera parte. Aquí va a desarrollar su primeras funciones en Python.

PRIMERA PARTE: REPASO DE CONCEPTOS

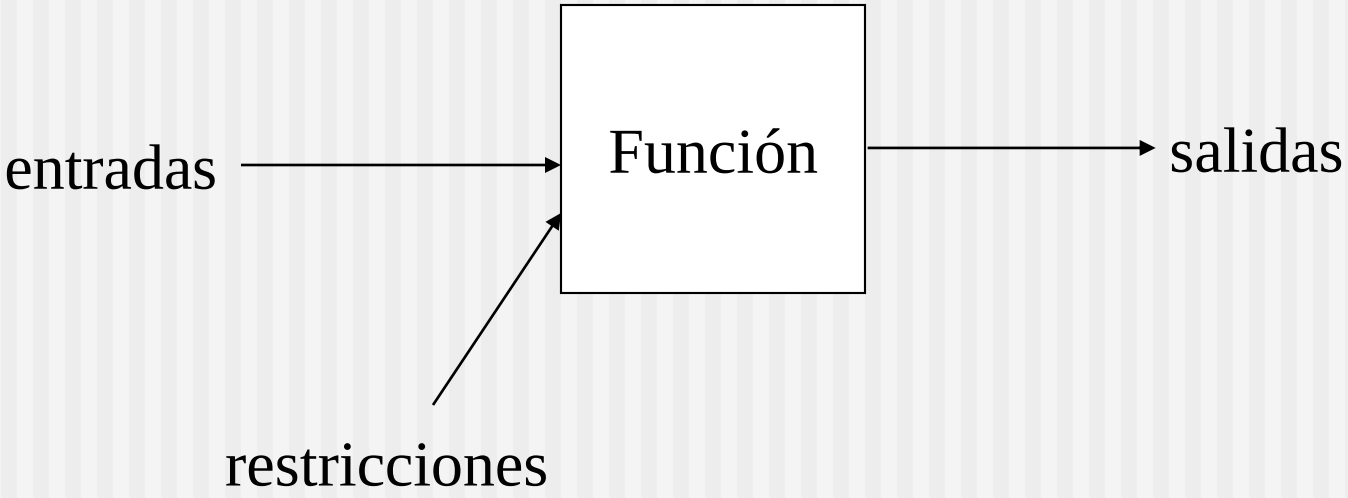
Instrucciones:

- Estudiar los conceptos acerca de funciones.
- Escribir y estudiar códigos de los ejemplos.
- Responder las preguntas.
- Hasta que termine esta parte conceptual continúe con la siguiente. Estos conceptos se usarán durante todo el curso.

FUNCIONES EN PROGRAMACIÓN

- Generalmente un programa está construido por una o más funciones, llamadas también subprogramas o procedimientos.
- Función: conjunto de instrucciones que realizan una tarea específica.
- Cada función contribuye con una parte de la solución y entre todas las funciones se obtiene la solución completa
- Hay dos tipos de funciones:
  - Las preconstruídas: ya están hechas, nosotros solo las usamos
  - Las desarrolladas por el programador: nosotros las desarrollamos

- Modelo general de una función



- Note que es el mismo modelo de un programa: hay datos de entrada a un proceso para obtener los resultados o salidas

Empecemos con una función simple.

Escriba en modo programa este ejemplo.

La función sucesor recibe un entero, le suma 1 y retorna ese resultado, el cual es asignado a una variable para luego imprimirlo y calcular su doble.

El símbolo de # se usa para poner comentarios en los programas. Son comentarios a nivel de línea: pueden estar en líneas separadas o posterior a una instrucción en la misma línea

```
# Función sucesor
def sucesor(num):
    suc=num+1
    return suc

# programa principal: leer datos, llamar a la función, imprimir
n=int(input("Dé un número: "))
s=sucesor(n)          # el resultado de la función se guarda en s
print ("El sucesor de" , n, "es", s)
print ("El doble de" , s, "es", s*2)
```

PROBAR FUNCIONES

Hay dos maneras de probar las funciones:

1- Hacemos la función y un programa principal.
En el programa principal generalmente hacemos estos pasos:

- pedimos al usuario los datos que se van a enviar a la función
- luego llamamos a la función con esos datos
- por último desplegamos los resultados retornados por la función

Esto se muestra en el ejemplo de la función sucesor.
Otra manera: en el programa principal en lugar de pedir datos al usuarios la llamamos con datos constantes y desplegamos los resultados retornados.

2- Hacemos la función solamente.
Luego de guardar la función vamos al modo comando y la ejecutamos desde ahí enviando los valores necesarios. En este modo los valores retornados son desplegados en forma automática. Pruebe este ejemplo:

```
>>> sucesor(10)
```

Pruebe la función sucesor con varios datos.

Elimine de la función la instrucción return suc y en su lugar ponga print(suc). Haga pruebas con diferentes datos de entrada.

¿ Qué sucede ? Analice la situación.

Ponga nuevamente return suc en lugar de print(suc). Después de hacer cualquier cambio en un programa éste debe ser probado nuevamente para verificar su correctitud.

¿ Funciona correctamente este programa ?

CONSIDERE ESTOS ASPECTOS EN EL DESARROLLO DE FUNCIONES

- Las definiciones de funciones NO alteran el flujo de ejecución del programa
- Las funciones se ejecutan cuando son llamadas
- La ejecución de funciones SI altera el flujo de ejecución del programa
  - ¿Cómo se altera este flujo de ejecución cuando se llama a una función?

- Al llamar a una función:
  - El flujo de ejecución se traslada a la función
  - Se ejecutan las instrucciones de la función
  - Cuando la función termina:
    - ✓ Se borran todas las variables locales de la función, es decir, aquellas que se usaron como parámetros y las variables a las cuales se les asignó un valor en la función
    - ✓ El flujo de ejecución del programa regresa al punto donde se hizo la llamada con los resultados que la función retornó (por medio de la instrucción return)

- Definiciones formales
  - Argumentos: son los valores que se envían a una función
  - Parámetros: la función recibe esos valores en variables llamadas parámetros
  - Para efectos prácticos: los conceptos argumentos y parámetros se usan indistintamente

- Variables locales a la función
  - Los parámetros definidos en la función son variables locales
  - También son variables locales las que se crean dentro de la función (en estatutos de asignación)
  - Las variables locales existen solo dentro de la función
  - No pueden ser usadas fuera de ella
    - ✓ Cuando la función termina la ejecución las variables que se crearon en esa función desaparecen, por tanto no pueden ser usadas después
  - Los valores calculados en las funciones que se van a usar posteriormente a su finalización deben ser retornados usando el estatuto **return**

- En el ejemplo de la función sucesor ponga esta instrucción al final del programa principal:

```
print(suc, num)
```

¿ Qué sucede ? Analice la situación.

Luego elimine esta instrucción de la función.

Recuerde: Después de hacer cualquier cambio en un programa éste debe ser probado nuevamente para verificar su correctitud.

¿ Funciona correctamente este programa ?

Escriba este ejemplo en modo programa: la función suma recibe dos argumentos, los suma e imprime el resultado

```
# definición de funciones
def suma(n1,n2):
        resul=n1+n2
        print(resul)

# programa principal
num1=float(input("Dé un número: "))
num2=float(input("Dé otro número: "))
suma(num1,num2)
print("Fin")
```

Cuando termina la ejecución de la función, las variables n1, n2 y resul desaparecen de la memoria porque son variables locales, es decir, variables que se pueden usar solo en la función.

- Se le solicita que agregue dos instrucciones en el programa principal luego de llamar a la función: asignar a una variable el resultado de la función dividido entre 2, imprimir esta división.

¿ Lo podemos hacer tal como esta el programa ?

Analice su respuesta.

¿ Qué propone para llevar a cabo la solicitud ?

Haga los cambios necesarios en el programa.

ESTRUCTURA BÁSICA RECOMENDADA PARA PROGRAMAS PYTHON

- Aunque Python no exige una estructura para un programa, para efectos de orden podemos usar esta
  - Comentarios del encabezado del programa (lo que hace el programa, entradas, salidas, autor, fecha de creación)
  - Estatutos import (para usar módulos)
  - Definir clases (lo veremos al final del curso)
  - Definir funciones
  - Programa principal

```
# los programas inician la ejecución
# en el programa principal: aquellas
# instrucciones que no pertenecen
# a definiciones de funciones.
```

VALORES RETORNADOS POR LAS FUNCIONES

- Al terminar la ejecución de una función las variables que se crearon en la función (variables locales a la función) desaparecen de la memoria por lo tanto no se pueden usar en el resto del programa
- Aquellos resultados obtenidos por las funciones que vayamos a ocupar posteriormente a su finalización, deben ser retornados (ver instrucción return)
- Los valores calculados dentro de una función que son retornados los podemos usar posteriormente en el programa
  - Asignándolos a una variable o
  - Usándolos directamente en una expresión
- Los valores calculados dentro de una función pero que no son retornados por la función no pueden ser usados luego de llamar a la función

- La función sucesor estudiada anteriormente retorna el VALOR que tiene la variable suc, luego ese valor puede ser usado fuera de la función. Note que se retorna el VALOR que contiene la variable.

```
# Función sucesor
def sucesor(num):
    suc=num+1
    return suc    # retorna el valor que contiene la variable suc

# programa principal: leer datos, llamar a la función, imprimir
n=int(input("Dé un número: "))
s=sucesor(n)    # el resultado de la función sucesor se guarda en una variable
                # para ser usado posteriormente
print ("El sucesor de" , n, "es", s)
print ("El doble de" , s, "es", s*2)
```

- Escriba en modo programa este ejemplo.

```
def calcular_raiz_cuadrada(n):
    ''' Comentarios multilínea en Python: pueden abarcar varias líneas. Inician y terminan con tres comillas: dobles o simples'''
    resul=n ** 0.5      # la raíz cuadrada se puede calcular
                        # elevando a 1/2
    return resul    # retorna el resultado

n=float(input("Calcular raíz cuadrada al número: "))
print (calcular_raiz_cuadrada(n))
```

ESTATUTO return

- Se usa en funciones. Cuando esta instrucción se ejecuta suceden dos eventos:
  - Retorna resultados que pueden ser usados posteriormente en el programa
  - La función termina inmediatamente

Forma general de uso:

return resultado1, resultado2, ...

- Cada resultado es una expresión que se convierte en un valor retornado
- Si en el return no se especifican valores, automáticamente retorna el valor especial llamado None

Escriba en modo comando este ejemplo:

- El resultado retornado por la función abs se guarda en una variable que es usada posteriormente en print

```
>>> a=5
>>> r=abs(a)
>>> print ("El valor absoluto de", a," es ",r)
```

- El resultado retornado por la función abs se usa directamente en una instrucción, en este caso el print

```
>>> b=-10
>>> print ("El valor absoluto de", b," es ",abs(b))
```

- Una función termina bajo dos condiciones
  - Después de ejecutar una instrucción return
  - Cuando no existan mas instrucciones que ejecutar en la función
- El valor None también es retornado cuando la función termina porque ya no tiene más instrucciones que ejecutar

- Los valores de entrada de una función pueden recibirse por medio de parámetros o se pueden leer directamente en la función.

- Escriba en modo programa esta función para convertir grados celsius a fahreinheit. Los datos de entrada no se van a recibir como parámetros, se van a leer en la función.

```
def convertir_grados_celsius_a_fahreinheit():
    celsius = float(input("Grados celsius: "))
    fahreinheit = 9 / 5 * celsius + 32
    print("En fahreinheit:", fahreinheit)
```

Pruebe la función de estas dos manera:

1)

```
>>> convertir_grados_celsius_a_fahreinheit()
```

2)

```
>>> a = convertir_grados_celsius_a_fahreinheit()
>>> print(a)
```

¿Por qué obtenemos este valor None?

USO DE MÓDULOS EN PYTHON

- Un módulo es un programa que contiene un conjunto de funciones y otros elementos relacionados para facilitar el desarrollo de programas ya que permite reutilizar esas que ya tiene construídas. El módulo puede tener:
  - ✓ Funciones
  - ✓ Definiciones de diversos objetos como las clases, métodos, constantes
- Python provee un conjunto de módulos ya preconstruídos
- Cada módulo tiene un nombre
- import: instrucción para importar módulos. Luego de la importación el programa puede usar todos los elementos del módulo.

- ***Ejemplos con el módulo math:*** contiene funciones matemáticas. Revise otras formas de hacer el import.

```
>>> import math
>>> math.sqrt(25)
5.0
```

Importa todo el módulo
Usar función del módulo:
nombre_módulo.nombre_función

```
>>> import math as m
>>> m.sqrt(25)
5.0
```

Cambiar en este programa el nombre del módulo

```
>>> from math import sqrt
>>> sqrt(25)
5.0
```

Importa solo la función sqrt
En estos tipos de import no usa el nombre del modulo para llamar a la función

```
>>> from math import *
>>> sqrt(25)
5.0
```

Otra forma de importar todo el módulo

```
>>> from math import sqrt as raiz
>>> raiz(25)
5.0
```

Importa la función sqrt y le cambia el nombre en este programa

- Listar los elementos de un módulo

```
>>> import math
>>> dir(math)
```

- Documentación de los elementos de un módulo

```
>>> import math
>>> help(math)
```

- Documentación de un elemento específico

```
>>> import math
>>> help(math.sqrt)
    Help on built-in function sqrt in module math:
    sqrt(...)
        sqrt(x)

        Return the square root of x.
```

SEGUNDA PARTE: DESARROLLO DE FUNCIONES

Instrucciones

- Haga solo un programa fuente (archivo fuente con el nombre práctica1_su_nombre.py) que contenga solamente la definición de todas las funciones usando los nombres indicados en cada una de ellas.
- Pruebe las funciones desde el modo comando.
- **USAR EL MATERIAL VISTO A LA FECHA. No se permite el uso de tipos de datos de secuencias (listas, strings, tuplas), ni diccionarios ni conjuntos. El objetivo es desarrollar algoritmos usando datos numéricos.**
- Algunos de los ejercicios ya fueron hechos, lo que debe hacer es estructurarlos como funciones.
- Envíe esta segunda parte de la práctica al tecDigital / DOCUMENTOS / EJEMPLOS_DE_PROGRAMAS

Función 1: componer_numero

Desarrolle una función que reciba tres dígitos en este orden: unidades, decenas y centenas y retorne una sola variable con el número que representan tal como muestra el ejemplo del funcionamiento.

Al indicar que la **función recibe datos** nos referimos a que la función es llamada con argumentos los cuales se reciben en parámetros.

Al indicar que la **función retorna los resultados** nos referimos a que la función debe usar **return** para ello, no imprime resultados dentro de la función (no usa **print**).

Ejemplo del funcionamiento:

```
>>> componer_numero(4, 5, 6)
654
```

NOTE QUE AL LLAMAR A UNA FUNCIÓN EN EL MODO COMANDO, LOS VALORES RETORNADOS SE MUESTRAN AUTOMATICAMENTE EN LA PANTALLA

Función 2: descomponer_numero

Desarrolle una función que reciba un número natural de 3 dígitos e imprima su descomposición en unidades, decenas y centenas tal como muestra el ejemplo del funcionamiento.

Al indicar que la **función imprime los resultados** nos referimos a que debe usar **print** dentro de la misma.

Ejemplo del funcionamiento:

```
>>> descomponer_numero(482)
Unidades en el numero: 2
Decenas en el numero: 8
Centenas en el numero: 4
```

Función 3: al_reves

Desarrolle una función que recibe un número natural de 4 dígitos y retorne como resultado una sola variable en la cual el número quede al revés.

Ejemplos del funcionamiento:

```
>>> al_reves( 9805)
5089

>>> al_reves(1234)
4321
```

Función 4: tiempo

Desarrolle una función que lea una cantidad de segundos y calcule lo que representa en: horas, minutos, segundos (los restantes). Imprima los resultados dentro de la función. No retorne datos
Al indicar que la **función va a leer los datos** nos referimos a que no tiene parámetros, es decir, no van a enviarle datos, en su lugar debe usar **input** en la función para obtener los datos de entrada. Los datos son enteros por tanto también use la función **int** para pasar el dato leído que es de tipo string a un valor entero.

Ejemplo del funcionamiento:

```
>>> tiempo()
Cantidad de segundos: 7405

Representan:
                    2 horas
                    3 minutos
                    25 segundos
```

Función 5: reforestar

Desarrolle una función para calcular y retornar la cantidad de pinos, eucaliptos y cedros que se tendrán que sembrar en un bosque para su reforestación. Recibe un número que indica la cantidad de hectáreas a reforestar. Para hacer los cálculos tenga en cuenta estas consideraciones:

- una hectárea equivale a 10 mil metros cuadrados.
- en 10 metros cuadrados caben 8 pinos
- en 15 metros cuadrados caben 15 eucaliptos
- en 18 metros cuadrados caben 10 cedros.

La superficie del bosque se reforestará según se indica a continuación:

Superficie del bosque	Tipo de árbol a sembrar
el 50 %	pino
el 30 %	eucalipto
el 20 %	cedro

Ejemplos del funcionamiento (retorne solo la parte entera de los resultados):

```
>>> reforestar (2)  #AL RETORNAR 2 O MAS VALORES LOS VEREMOS ENTRE PARENTESIS (TUPLA)
(8000, 6000, 2222)
```

Esta función retorna 3 valores. Pruebe con estas formas de asignar a variables los resultados retornados.

```
>>> resultado = reforestar(2)
>>> print(resultado)

>>> pinos, eucaliptos, cedros = reforestar(2)
>>> print(pinos, eucaliptos, cedros)
```

Función 6: tiempo_total

Un proceso de transporte tiene dos etapas: la marítima y la terrestre. El tiempo que toma cada etapa está expresado como un número de 6 dígitos con tres segmentos: hhmmss. hh son las horas y van de 00 a 99, mm son los minutos y van de 00 a 59, ss son los segundos y van de 00 a 59.

Desarrolle una función que sume los tiempos de cada etapa para conocer el tiempo total que dura el proceso completo de transporte. La función debe leer dos datos (no los recibe como parámetros, los lee en la función):

- Tiempo que duró la parte marítima (hhmmss)
- Tiempo que duró la parte terrestre (hhmmss)

Considere que 1 hora tiene 60 minutos y 1 minuto tiene 60 segundos.

Debe imprimir (no retornar) una sola variable que contenga el resultado.

Ejemplo del funcionamiento:

```
>>> tiempo_total()
Tiempo que duró la parte marítima: 115530
Tiempo que duró la parte terrestre: 101545
Tiempo total: 221115
```

Función 7: sumar_y_multiplicar_digitos

Desarrolle una función que reciba un número natural de 4 dígitos y retorne dos valores: la suma de sus dígitos y el producto de sus dígitos.

Ejemplo del funcionamiento:

```
>>> sumar_y_multiplicar_digitos(5813)
(17, 120)
```

Los resultados se calcularon así:

5 + 8 + 1 + 3 = 17

5 * 8 * 1 * 3 = 120

Función 8: tabla_de_multiplicar

Desarrolle una función para representar la tabla de multiplicar de un número según se muestra en el ejemplo del funcionamiento. En la función se lee (NO RECIBE DATOS, LOS VALORES LOS LEE DENTRO DE LA FUNCIÓN CON INPUT) el número de la tabla y en la función se imprimen los resultados.

Ejemplo del funcionamiento:

```
>>> tabla_de_multiplicar()
Tabla de multiplicar del número: 3
3 x 0 = 0
3 x 1 = 1
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
```

Función 9: fecha

Desarrolle una función que reciba un número natural que contiene exactamente 8 dígitos representando una fecha de la siguiente manera: los 2 dígitos de más a la izquierda representan el día, los siguientes 2 dígitos representan el mes y los siguientes 4 dígitos representan el año.

La función toma ese valor y debe formar y retornar otra variable numérica que contenga exactamente 6 dígitos representado esa fecha con otro formato de la siguiente manera: los 2 dígitos de más a la izquierda representan el mes, los siguientes 2 dígitos representan el día y los siguientes 2 dígitos representan el año (quitar 2 dígitos de la izquierda).

Ejemplo del funcionamiento:

```
>>> fecha(25122019)
122519
```

Función 10: desglosar_moneda

Desarrolle una función que lea una cantidad de colones e imprima su desglose de billetes: la cantidad mínima de billetes que se deben dar para completar la cantidad solicitada. Esta cantidad es un múltiplo de 5000. Hay billetes de 50000, 20000, 10000 y 5000 colones. Los cálculos se hacen según las denominaciones de billetes, empezando por el billete de más alto valor y llegando al menor. Imprima los resultados según el ejemplo del funcionamiento:

```
>>> desglosar_moneda()
Cantidad de colones: 155000
Desglose de billetes:
        2 de 50000          100000
        2 de 20000           40000
        1 de 10000           10000
        1 de 5000             5000
```