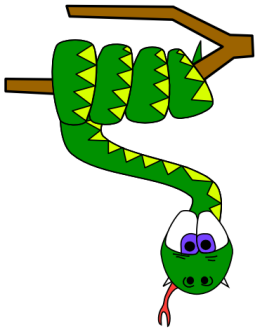

Programación en Python

Sesión 3

Funciones

Víctor Melchor

Contenido

- Funciones
  - Conceptos
  - Objetivos
  - Modularización
  - Ejemplos
  - Ejercicios

Abstracción

- Técnica de programación que nos permite pensar un problema en **diversos niveles**
  - Cuando pensamos en un problema macroscópicamente, no estamos preocupados con los detalles
- ***Dividir para conquistar***:
  - Un problema es dividido en diversos sub-problemas
  - Las soluciones de los **sub-problemas** se combinan en una solución del problema mayor

Programación Estructurada

- Usa el principio de “Dividir para Conquistar”
- Los programas se dividen en **sub-programas**
  - Cada sub-programa es llamado por medio de un **identificador** y una lista de **parámetros** de **entrada**
    - Permite especificar como un problema puede ser resuelto *en general*
    - El mismo sub-programa puede ser invocado para resolver diversos problemas de la **misma naturaleza** pero con valores específicos diferentes.

Funciones

- En Python, los sub-programas tienen el nombre de ***funciones***
- Formato general:

```
def nombre (arg, arg, ... arg):
        comando
        . . .
        comando
```

- Donde:
  - **nombre** es el nombre de la función
  - **args** son especificaciones de argumentos de la función
    - Una función puede tener 0, 1 o mas argumentos
  - **comandos** contienen las instrucciones a ser ejecutadas cuando la función es invocada

Funciones sin Parámetros

La siguiente función no tiene parámetros. Sin embargo, en la llamada debe usar obligatoriamente los paréntesis.

```
import datetime

def imprime_fecha_actual():
    fecha_actual = datetime.datetime.now().date()
    print(fecha_actual)

imprime_fecha_actual()
```

Funciones con un Parámetro

- Procedimiento

```
def imprime_mensaje(text):
    print(text)

imprime_mensaje("Bienvenido a Python")
```

Funciones y Procedimientos

- Procedimiento

```
def fib(n):
    #Escribe la serie de Fibonacci hasta n
    a,b=0,1
    print(a)
    while b<n:
        print(b)
        a , b = b , a+b

fib(8)
```

Resultado de Funciones

- Una función típicamente calcula **uno** o **más valores**
- Para indicar el valor a ser devuelto como el resultado de la función, se usa el comando return, que tiene el formato

  return *expresion*

  – donde la *expresión* es opcional *y* designa el valor a ser retornado

Resultado de Funciones

- Al encontrar el comando **return**, la función termina inmediatamente y el control del programa **vuelve al punto** donde la función fue llamada
- Si una función llega a su fin sin que ningún valor de retorno haya sido especificado, el valor de retorno es **None**

Funciones

- **`return`** termina la función retornando un valor

- El valor predeterminado de return es **`None`**

- Si la función llegara al fin sin el uso explícito del **`return`**, entonces también se retornará el valor **`None`**

Funciones que retornan un Valor

- Uso de return

```
def suma_numeros(num1, num2):
    return num1 + num2

result = suma_numeros(10, 5)
print(result)
```

Funciones que retornan múltiples Valores

- Uso de return

```
import datetime

def retorna_fechahora_actual():
    fechahora_actual = datetime.datetime.now()
    fecha_actual = fechahora_actual.date()
    hora_actual = fechahora_actual.time()
    return (fecha_actual, hora_actual)

fa, ha = retorna_fechahora_actual()
print(fa)
print(ha)
```

Recursividad

- Es un principio muy **poderoso** para la construcción de algoritmos
- La solución de un problema se **divide** en
  - Casos simples:
    - Son aquellos que pueden ser resueltos **trivialmente**
  - Casos generales:
    - Son aquellos que pueden ser resueltos **proponiendo soluciones** de casos más simples

Funciones Recursivas

- Algoritmos recursivos donde la solución de los casos genéricos requieren **llamadas a la propia función.**
- Ejemplo: Secuencia de Fibonacci
  - El **primer** y el **segundo** término son **0** y **1**, respectivamente
  - El ***i*-ésimo** término es la suma del **(*i*-1)-ésimo** y el **(*i*-2)-ésimo** término

Funciones Recursivas

- Ejemplo: Factorial
  - **Factorial(1) = 1**
  - **Factorial(i) = i * Factorial(i – 1)**

```
def facRec(num):
    #retorna el factorial de num
    if num==1:
        return 1
    else:
        return num*facRec(num-1)

y=facRec(6)
print(y)
```

Recursividad

Ejemplo

```
def fibRec(n):
    #retorna el termino fibonacci de n
    if n==1:
        return 0
    elif n==2:
        return 1
    else:
        return fibRec(n-1)+fibRec(n-2)

for i in range(1,8):
    print(fibRec(i))
```

Variables Locales y Globales

- Variables **definidas en funciones** son ***locales***, esto es, solo pueden ser usadas en las funciones en las que fueron definidas.
- Variables definidas **fuera de las funciones** son conocidas como variables **globales**
  - En una función **se puede** leer el contenido de una variable global
  - Para **alterar una variable global**, se debe declararla en el cuerpo de la función como **global**

Ejemplo

```
def suma_numeros(num1, num2):
    result = num1 + num2
    return result

print(suma_numeros(10, 5))
print(result)
```

```
15
Traceback (most recent call last):
  File "C:\DATOS\VICTOR\CTIC\Tecnologia\Proc
unciones\fun15.py", line 9, in <module>
    print(result)
NameError: name 'result' is not defined
>>>
```

Ejemplo

```
nombre = "Charlie" # Global variable

def suma_numeros(num1, num2):
    saludo = "Hola, " + nombre # saludo es una variable local
    suma = num1 + num2 # suma es variable local
    result = saludo + " la suma es " + str(suma)
    return result

print(suma_numeros(10, 5))
print(result)
```

```
Hola, Charlie la suma es 15
Traceback (most recent call last):
  File "C:/DATOS/VICTOR/CTIC/Tecnologia/
unciones/fun16.py", line 12, in <module>
    print(result)
NameError: name 'result' is not defined
>>>
```

Ejemplo

```
result = 100

def suma_numeros(num1, num2):
    result = num1 + num2
    return result

print(suma_numeros(10, 5))
print(result)
```

```
 RESTART: C:/DATOS/VICTOR/
 funciones/fun17.py
15
100
>>>
```

Ejemplo

```
materia = "programacion"

def cambiar_materia(nuevo_valor):
    """Cambia el valor para la variable global materia"""
    global materia # Declara materia como global
    materia = nuevo_valor

print(materia)
cambiar_materia("machine learning")
print(materia)
```

```
programacion
machine learning
>>> 
```

Argumentos de funciones

- **Argumentos (o parámetros)** son variables que reciben valores iniciales en la **llamada** de la función.
- Esas variables son ***locales***
- Si una función define ***n*** argumentos, en su llamada **debe incluir valores** para todos ellos
  - **Excepción**: los argumentos con valores **predeterminados**

Ejemplo

```
def f(x):
    return x*x
```

```
>>> print(f(10))
100
>>> print(x)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>> print(f())
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    print(f())
TypeError: f() missing 1 required positional argument: 'x
>>>
```

Argumentos *predeterminados*

- Es posible dar valores ***predeterminados*** a argumentos
  - Si el llamador no especifica valores para esos argumentos, son usados los **predeterminados**
- Formato:

  def nombreFuncion (arg1=default1, ..., argN=defaultN)
- Si solo algunos de los argumentos tienen valores predeterminados, estos deben ser los ***últimos***

Ejemplo

```
def f(nombre,saludo="Hola",exclamacion="!!"):
    return saludo+", "+nombre+exclamacion

print(f("Juan"))
```

```
Hola, Juan!!
>>> print(f("Luis","Exitos"))
Exitos, Luis!!
>>> print(f("Maria","Hasta luego","..."))
Hasta luego, Maria...
>>>
```

Argumentos

```
def muestra_info(obligatorio,nombre="Juan",edad=20):
    print("obligatorio:%s\nnombre: %s\nedad: %d"%(obligatorio,nombre,edad))

muestra_info("Java")
```

```
obligatorio:Java
nombre: Juan
edad: 20
>>> muestra_info("Java",10)
obligatorio:Java
nombre: 10
edad: 20
>>> muestra_info("Java",edad=10)
obligatorio:Java
nombre: Juan
edad: 10
>>>
```

EJERCICIOS

Ejercicios

1. Hacer una función que reciba como parámetro un número entero y retorne el factorial de ese número (no usar recursividad).
2. Hacer una función que reciba tres argumentos, y que retorne la suma de esos tres argumentos.
3. Elabora una función llamada ***sumaImpuesto***. La función posee dos parámetros :
   a) tasaImpuesto, que es el porcentaje de impuesto sobre ventas
   b) costo, que es el costo de un item antes del impuesto.

   La función retorna el valor de costo alterado para incluir el impuesto sobre ventas.

Ejercicios

4. Elabore un programa que convierta de la notación de 24 horas para la notación de 12 horas. Por ejemplo, el programa debe convertir 14:25 en 2:25 P.M; 6:44 en 6:44 A.M. La entrada está dada en dos enteros. El programa debe leer varias entradas y llamar a una función para convertirlas y en seguida imprimir la salida.
5. Hacer una función que reciba un argumento entero. La función retorna el valor del carácter ‘P’, si su argumento fuera positivo, y ‘N’, si su argumento es cero o negativo.
6. Realice una función que retorne el reverso de un número entero ingresado por el usuario. Por ejemplo: 127 -> 721.

Ejercicios

7. Elabora una función que informe la cantidad de dígitos de un determinado número entero ingresado por el usuario.
8. Elabora un programa que permita al usuario digitar su nombre y en seguida el programa llama a una función que retorna el nombre del usuario de atrás hacia adelante utilizando solamente letras mayúsculas. Sugerencia: Considere que al informar el nombre el usuario puede digitar letras mayúsculas o minúsculas.

Ejercicios

9. Elabora un programa que solicite la fecha de nacimiento (dd/mm/aaaa) del usuario e imprima la fecha con el nombre del mes completo. El programa debe llamar una función que retorna el mes convertido. **Ejemplo:**
   - Entrada - Fecha de Nacimiento: 29/10/1993
   - Salida - Usted nació el 29 de Octubre de 1993.

Ejercicios

10. Considere la siguiente fórmula para calcular el mcd (máximo común divisor) de dos números enteros positivos:
   - mcd(a, b) = b, si **b** divide a **a** (o sea, a%b == 0)
   - mcd(a, b) = mcd(b, a%b), en caso contrario

   Escriba una función en Python tal que, dados dos números, retorne el máximo común divisor entre ellos. **Usar recursividad**.