TEMA II

DESARROLLO DE APLICACIONES DE PROPÓSITO GENERAL

"Un sueño no se hace realidad por arte de magia, necesita sudor, determinación y trabajo duro"
Colin Powell

3. CONTROL DE FLUJO

Condicionales

Alternativas simples

```
if numero<0:
    print("Número es negativo")
```

Alternativas dobles

```
if numero<0:
    print("Número es negativo")
else:
    print("Número es positivo")
```

Alternativas múltiples

```
if numero>0:
    print("Número es negativo")
elif numero<0:
    print("Número es positivo")
else:
```

```
        print("Número es cero")
```

Expresión reducida del if

```
>>> lang="es"
>>> saludo = 'HOLA' if lang=='es' else 'HI'
>>> saludo
'HOLA'
```

Bucles

While

while(condicion):

```
año = 2001
while año <= 2017:
    print ("Informes del Año", año)
    año += 1
else:
        print ("Hemos terminado")
```

For

for variable in elemento a recorrer:

Donde el elemento a recorrer puede ser una cadena, una lista, una tupla, un diccionario, un rango….

```
for i in range(1,10):
   print (i)
else:
        print ("Hemos terminado")
```

En este caso el bucle empieza en 1 y termina en 9

```
for i in range(5)
    print (i)
```

En este caso empezamos en 0 y terminamos en 4

```
for i in range(4,10,2): #Secuencia del 4 al 9 , de 2 en 2
    print(i)
```

En ambos casos podemos prescindir de la sentencia else, el resultado final sería el mismo excepto si utilizamos la sentencia break.

Break

Termina la ejecución del bucle, además no ejecuta el bloque de instrucciones indicado por la parte else.

Continue

Deja de ejecutar las restantes instrucciones del bucle y vuelve a iterar.

Recorriendo varias secuencias : función zip()

Con la instrucción for podemos ejecutar más de una secuencia, utilizando la función zip. Esta función crea una secuencia donde cada elemento es una tupla

```
>>> for x,y in zip(range(1,5),["ana","juan","pepe"]):
...     print(x,y)
1 ana
2 juan
3 pepe
```

Marca el final del bucle la “lista” más pequeña

Funciones

Para definir una función se antepone la palabra def a su declaración

```
>>> def factorial(n):
...     """Calcula el factorial de un número"""
...     resultado = 1
...     for i in range(1,n+1):
...         resultado*=i
...     return resultado
```

Para utilizarla, se la llama pasando los parámetros

Las variables declaradas dentro de la función son locales a la misma, podemos declarar una variable global para que pueda ser utilizada fuera

```
>>> def operar(a,b):
...     global suma
...     suma = a + b
...     resta = a - b
...     print(suma,resta)
...
>>> operar(4,5)
9 -1
>>> resta
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'resta' is not defined
>>> suma
9
```

Podemos definir variables globales, que serán visibles en todo el módulo. Se recomienda declararlas en mayúsculas:

```
>>> PI = 3.1415
>>> def area(radio):
...    return PI*radio**2
...
>>> area(2)
12.566
```

Paso de parámetros

En Python el paso de parámetros es siempre por referencia. El lenguaje no trabaja con el concepto de variables sino objetos y referencias. Al realizar la asignación a = 1 no se dice que "a contiene el valor 1" sino que "a referencia a 1". Así, en comparación con otros lenguajes, podría decirse que en Python los parámetros siempre se pasan por referencia.

Parámetros keyword (parámetros por defecto)

Los parámetros keyword son aquellos donde se indican el nombre del parámetro y su valor, por lo tanto no es necesario que tengan la misma posición. Al definir una función o al llamarla, hay que indicar primero los argumentos "fijos" y a continuación los argumentos con valor por defecto (keyword).

```
>>> def operar(n1,n2,operador='+',respuesta='El resultado es '):
...  if operador=="+":
...    return respuesta+str(n1+n2)
...  elif operador=="-":
...    return respuesta+str(n1-n2)
...  else:
...    return "Error"
```

Son posibles varias formas de llamadas

```
>>> operar(5,7) # dos parámetros posicionales
>>> operar(4,6,respuesta="La suma es")  # dos parámetros posicionales y uno keyword
>>> operar(4,6,respuesta="La resta es",operador="-")    # dos parámetros posicionales y dos keyword
```

Argumentos arbitrarios (*args y **kwargs)

Para indicar un número indefinido de argumentos "fijos" al definir una función, utilizamos el símbolo *:

```
>>> def sumar(*args):
...   resultado=0
...   for i in args:
...     resultado+=i
...   return resultado
...
>>> sumar(2)
2
>>> sumar(2,3,4)
9
```

Para indicar un número indefinido de argumentos keyword al definir una función, utilizamos el símbolo **:

```
def funcion3(semilla=1,**kwargs):
    resultado=semilla
    for x in kwargs.values():
        resultado+=x
    print(resultado)

funcion3()
funcion3(semilla=3,n=4,b=5)
```

Devolver múltiples resultados

La instrucción return puede devolver cualquier tipo de resultados, por lo tanto, es fácil devolver múltiples datos guardados en una lista o en un diccionario. Veamos un ejemplo en que devolvemos los datos en una tupla:

```
>>> def operar(n1,n2):
...   return (n1+n2,n1-n2,n1*n2)

>>> suma,resta,producto = operar(5,2)
```