Python para Análisis Numérico

MAT-156 Análisis Numérico

Python es...

- un lenguaje de programación interpretado, orientado a objetos, lenguaje de alto nivel.
- un lenguaje simple y una sintaxis fácil de aprender
- fue el lenguaje de programación más popular el 2017

Comentarios

```
1
2   # Este es un comentario
3
4   '''
5   Este es un comentario en bloque
6   Cualquier linea entre el principio y el final
7   es parte de este comentario
8   '''
9
```

Lectura y Escritura

La función *print*, nos permite imprimir el contenido de una variable a la pantalla por consola.

Esta función nos permite modificar la forma de la salida que deseamos, siendo las dos estilizaciones más comunes la separación entre variables a imprimir, y el carácter de fin de línea.

```
>>> n = 12                      #Asignamos un valor numérico a "n"
>>> print(n)                    #Mostramos el valor de "n"
12                              #La salida de la función Print
>>> s = "hola mundo"            #Asignamos una cadena de texto a "s"
>>> print(s)
hola mundo                      #Se muestra el contenido de la variable  "s"
>>> s1 = 'hola mundo'           #Es equivalente el uso de comilla  simple o doble
>>> print(s1)
hola mundo
>>> s1
>>> 'hola mundo'                #Si no usamos la función print, se  muestra con comillas
>>> f = 1/3                     #Igual podemos guardar decimales en variables
>>> print(f)                    #La función print igual puede mostrar el valor
0.3333333333333333
>>> s1 = 'hola'
>>> s2 = 'como'
>>> s3 = 'estas'
>>> print(s1, s2, s3)           #Podemos imprimir varias variables a la vez
hola como estas                 #Se agrega un espacio automaticamente
>>> print(s1, s2, n)            #Funciona con variables de diferentes  tipos
hola como 12
>>> print(s1, s2, s3, sep='-')  #Podemos cambiar el tipo de separación
hola-como-estas
>>> print(s1, s2, s3, sep='-', end='') #Podemos cambiar el caracter de fin de línea
hola-como-estas>>>              #Para que no imprima un salto de línea
```

Leer Datos de Entrada

La función input, es la forma estándar de lectura de datos por teclado, los datos que ingresemos se reciben como una cadena de texto dentro de la variable.

Podemos mostrar una línea de texto antes de recibir la entrada si la ingresamos como argumento de la función.

```
>>> var = input()          #Le decimos al intérprete que lo ingresado se guardará en "var"
1998                       #Ingresamos el dato por teclado
>>> print(var)
1998                       #Verificamos que se ingresó correctamente
>>>var
'1998'                     #Al no formatear, obeservamos que es una cadena
>>> type(var)
<class 'str'>              #Confirmamos que "var" contiene una cadena
>>> nombre = input('ingrese su nombre: ')   #Cadena como parámetro
ingrese su nombre: Juan    #Escribimos el nombre al lado de la cadena impresa
>>> print(nombre)
```

Tipos de Datos

Dado que la entrada estándar siempre nos devolverá una cadena por teclado,muchas veces necesitaremos utilizar los datos como números enteros, o decimales. Para estas situaciones, debemos convertir el dato de tipo cadena a los tipos de datos que necesitemos utilizar, afortunadamente, Python ya tiene todas esas funciones pre implementadas.

```
>>> numero = int(input())          #La función int() convierte  cadenas a enteros
33                                 #Ingresamos el dato
>>> type(numero)
<class 'int'>              #Verificamos que efectivamente "numero" contiene un entero
>>> numero
33                         #Al no tener comillas, observamos que es un número
>>> decimal = float(input())      #Usamos la función "float()"
3.56                               #Para convertir cadenas a decimales
>>> type(decimal)
<class 'float'>            #Verificamos que efectivamente "decimal" contiene un real
>>> decimal
3.56
```

Estructuras de Condicionales - If

Es una estructura condicional, que recibe una expresión lógica que puede ser simple o compuesta y si se cumple, se ejecutan una serie de instrucciones.

En el caso de Python, a diferencia de otros lenguajes en los cuales las instrucciones a ejecutar dada la condición se engloban dentro de llaves, las líneas dentro de la condicional se definen en un nivel de indentación, para así expresar la jerarquía del código.

Para realizar comparaciones, se debe usar el operador == que es diferente al operador de asignación =.

```
>>> a = True              #Asignamos el valor verdad a "a"
>>> if a == True:         #"if" verifica si la condición se cumple
...     print('hola')     #Se usa un bloque de indentación para indicar
...                       #que la instrucción se ejecuta bajo la condición
hola                      #Salida del programa
>>> b = False
>>> if a == True and b == False: #Se pueden usar expresiones compuesta
...     print('hola 2')
...
hola 2
```

Estructuras Condicionales - Else

Es la contraparte de la condicional, permite definir un bloque de código a ejecutarse en caso de que la condición del if previo no se llegue a cumplir y se deba hacer algo en ese caso.

```
>>> a = True
>>> b = True
>>> if a == True and b == False:
...     print('hola')
... else:
...     print('adios')
...
adios
```

Estructuras Condicionales - Elif

Es una variación del Else pero en este caso, permite al igual que el if normal usar una condicional, vendría a ser el equivalente lógico de decir “Si no pasa esto, entonces, si es que pasa esto otro, hacer algo”.

```
>>> a, b = False, True                    #Este tipo de asignación
                                          #asigna: a = False y b = True
>>> if a == True and b == False:
...     print('hola')
... elif a == True and b == True:
...     print('adios')
... else:
...     print('bienvenido')
...
bienvenido
```

Estructuras Repetitivas - While

Es un bucle que depende de una condicional que al igual que en la sentencia If, puede ser simple o compuesta. El conjunto de instrucciones se repetirá mientras la condicional se cumpla, en el momento en que ya no se dé el caso, el bucle finaliza y se procede a ejecutar la siguiente instrucción fuera del bucle.

```
>>> n = 0
>>> while n < 4:
...     print(n)
...     n = n + 1
...
0
1
2
3
```

Estructuras Condicionales - For

Range

Es una función que permite generar un rango de valores de formas diferentes dependiendo de los argumentos que recibe. En el caso de que reciba un argumento n, el rango se genera desde 0 hasta n−1. En el caso de que reciba dos argumentos n y m, el rango se genera desde n hasta m − 1.

For

Es un bucle en el que podemos definir un mayor control en las condiciones iniciales, siendo la variable iteradora, y el rango de valores en los que se iterará, mediante la función range.

```
>>> for i in range(2, 6):
...     print(i)
...
2
3
4
5
```

Funciones - Argumentos

Es el conjunto de datos que recibirá la función y con los que trabajará localmente. Se definen en la cabecera de la función, entre paréntesis y luego del nombre de ésta.

```
def funcion(arg1, arg2, arg3, ...):
    #Instrucciones
```

Funciones - Retorno

Para devolver una o más variables al finalizar una función, debemos usar la instrucción return, al final de la ésta.

```
>>> def suma_rango(a, b):
...     res = 0
...     for i in range(a, b+1):    #Iteramos hasta b+1 porque
...         res = res + i          #Range corre hasta b-1
...     return res                 #Retornamos el resultado
...
>>> sum = suma_rango(1, 10)       #sum recibe la suma resultante
>>> print(sum)
55
```

Funciones Lambda

Son funciones que se pueden definir en una línea, que pueden o no tener nombre y tienen diversos usos por ejemplo al programar funciones matemáticas, o también al usar métodos de comparación complejos dentro de otras funciones.

```
>>> funcion = lambda  x, y : (x+2) * y
>>> print( funcion(2, 3) )
12
```

Listas

Una lista es una estructura de datos y un tipo de dato en python con características especiales. Lo especial de las listas en Python es que nos permiten almacenar cualquier tipo de valor como enteros, cadenas y hasta otras funciones; por ejemplo:

```
lista = [1, 2.5, 'MAT-156', [5,6] ,4]
```

Una lista es un arreglo de elementos donde podemos ingresar cualquier tipo de dato, para acceder a estos datos podemos hacer mediante un índice.

```
print lista[0] # 1
print lista[1] # 2.5
print lista[2] # MAT-156
print lista[3] # [5,6]
print lista[3][0] # 5
print lista[3][1] # 6
print lista[1:3] # [2.5, 'MAT-156']
print lista[1:6] # [2.5, 'MAT-156', [5, 6], 4]
print lista[1:6:2] # [2.5, [5, 6]]
```

Iterar en Listas

Una lista es un arreglo de elementos donde podemos ingresar cualquier tipo de dato, para acceder a estos datos podemos hacer mediante un índice.

```
for element in lista:
    print element
```

Listas - Métodos

- **append()** : Este método nos permite agregar nuevos elementos a una lista.
- **extend()**: Extend también nos permite agregar elementos dentro de una lista, pero a diferencia de append al momento de agregar una lista, cada elemento de esta lista se agrega como un elemento más dentro de la otra lista.
- **remove()**: El método remove va a remover un elemento que se le pase como parámentro de la lista a donde se le esté aplicando.
- **index()**: Index devuelve el número de índice del elemento que le pasemos por parámetro.
- **count():** Para saber cuántas veces un elemento de una lista se repite podemos utilizar el metodo count().
- **reverse()**: con el metodo reverse invertimos los elementos de una lista.

Derivadas e Integrales

SymPy

SymPy es una biblioteca Python para matemática simbólica. Su propósito es convertirse en un completo sistema de álgebra computacional que pueda competir directamente con alternativas comerciales manteniendo, a la vez, el código tan simple como sea posible para hacerlo extensible de manera fácil e integral. SymPy está escrito completamente en Python y no necesita usar otras bibliotecas.

Derivadas con SymPy

Podemos derivar casi cualquier expresión SymPy usando **diff(func, var)**.

Ejemplos:

Importamos la librería ***SymPy***

```
from sympy import *
x, y = symbols('x y')
init_printing(use_unicode=False)
```

Derivamos con respecto a ***x***

```
d = diff(x**2,x) # derivada de x^2
d
```

$$2x$$

Evaluar expresiones simbólicas

Para calcular el resultado de expresiones simbólicas con un determinado valor debemos usar el método ***subs()*** y el método ***evalf()*** de SymPy:

Del ejercicio anterior:

```
d.subs(x, 1).evalf() # evaluamos con x = 1
```

```
2.0
```

Integrales con SymPy

SymPy ofrece soporte para integrales definidas o indefinidas de funciones transcendentes elementales y de funciones especiales via ***integrate()***. Podemosintegrar funciones elementales de la siguiente forma:

```
x = symbols('x')
I = integrate(6*x**5, x)
I
```

$$x^6$$