Tema 0 - Programación básica con Python II

Curso de Física Computacional

M. en C. Gustavo Contreras Mayén

Contenido

Contenido

Contenido

Contenido

Contenido

La función input()

La función `input()` permite obtener texto escrito por teclado.

Al llegar a la función, el programa se detiene esperando que se escriba algo y se pulse la tecla Enter, como muestra el siguiente ejemplo:

```
>>> nombre = input("¿Cómo te llamas? ")
¿Cómo te llamas? Gustavo
>>> print("Mucho gusto en conocerte, ", nombre)
Mucho gusto en conocerte, Gustavo
```

La función input()

La función `input()` permite obtener texto escrito por teclado.

Al llegar a la función, el programa se detiene esperando que se escriba algo y se pulse la tecla Enter, como muestra el siguiente ejemplo:

```
>>> nombre = input("¿Cómo te llamas? ")
¿Cómo te llamas? Gustavo
>>> print("Mucho gusto en conocerte, ", nombre)
Mucho gusto en conocerte, Gustavo
```

La función input()

La función `input()` permite obtener texto escrito por teclado.

Al llegar a la función, el programa se detiene esperando que se escriba algo y se pulse la tecla Enter, como muestra el siguiente ejemplo:

```
>>> nombre = input("¿Cómo te llamas? ")
¿Cómo te llamas? Gustavo
>>> print("Mucho gusto en conocerte, ", nombre)
Mucho gusto en conocerte, Gustavo
```

```
>>> print a
```

La función input()

La función `input()` permite obtener texto escrito por teclado.

Al llegar a la función, el programa se detiene esperando que se escriba algo y se pulse la tecla Enter, como muestra el siguiente ejemplo:

```
>>> nombre = input("¿Cómo te llamas? ")
¿Cómo te llamas? Gustavo
>>> print("Mucho gusto en conocerte, ", nombre)
Mucho gusto en conocerte, Gustavo
```

```
>>> print a
2
```

La función input()

La función `input()` permite obtener texto escrito por teclado.

Al llegar a la función, el programa se detiene esperando que se escriba algo y se pulse la tecla Enter, como muestra el siguiente ejemplo:

```
>>> nombre = input("¿Cómo te llamas? ")
¿Cómo te llamas? Gustavo
>>> print("Mucho gusto en conocerte, ", nombre)
Mucho gusto en conocerte, Gustavo
```

```
>>> print a
2
```

```
>>> a
```

La función input()

La función `input()` permite obtener texto escrito por teclado.

Al llegar a la función, el programa se detiene esperando que se escriba algo y se pulse la tecla Enter, como muestra el siguiente ejemplo:

```
>>> nombre = input("¿Cómo te llamas? ")
¿Cómo te llamas? Gustavo
>>> print("Mucho gusto en conocerte, ", nombre)
Mucho gusto en conocerte, Gustavo
```

```
>>> print a
2
```

```
>>> a
'a'
```

La función input()

La función `input()` permite obtener texto escrito por teclado.

Al llegar a la función, el programa se detiene esperando que se escriba algo y se pulse la tecla Enter, como muestra el siguiente ejemplo:

```
>>> nombre = input("¿Cómo te llamas? ")
¿Cómo te llamas? Gustavo
>>> print("Mucho gusto en conocerte, ", nombre)
Mucho gusto en conocerte, Gustavo
```

```
>>> print a
2
```

```
>>> a
'a'
```

```
>>> type(a)
<type 'str'>
```

La función input()

La función `input()` permite obtener texto escrito por teclado.

Al llegar a la función, el programa se detiene esperando que se escriba algo y se pulse la tecla Enter, como muestra el siguiente ejemplo:

```
>>> nombre = input("¿Cómo te llamas? ")
¿Cómo te llamas? Gustavo
>>> print("Mucho gusto en conocerte, ", nombre)
Mucho gusto en conocerte, Gustavo
```

```
>>> print a
2
```

```
>>> a
'a'
```

```
>>> type(a)
<type 'str'>
```

La función input()

La función `input()` permite obtener texto escrito por teclado.

Al llegar a la función, el programa se detiene esperando que se escriba algo y se pulse la tecla Enter, como muestra el siguiente ejemplo:

```
>>> nombre = input("¿Cómo te llamas? ")
¿Cómo te llamas? Gustavo
>>> print("Mucho gusto en conocerte, ", nombre)
Mucho gusto en conocerte, Gustavo
```

```
>>> print a
2
```

```
>>> a
'a'
```

```
>>> type(a)
```

```
>>> print b, type(b)
```

```
>>> print b, type(b)
2 <type 'int'>
```

```
>>> print b, type(b)
2 <type 'int'>


>>> s=eval(raw_input("Ingrese s :"))
```

```
>>> print b, type(b)
2 <type 'int'>


>>> s=eval(raw_input("Ingrese s :"))
Ingrese s: 2*3
```

```
>>> print b, type(b)
2 <type 'int'>

>>> s=eval(raw_input("Ingrese s :"))
Ingrese s: 2*3

>>> print s, type(s)
```

```
>>> print b, type(b)
2 <type 'int'>

>>> s=eval(raw_input("Ingrese s :"))
Ingrese s: 2*3

>>> print s, type(s)
6 <type 'int'>
```

```
>>> print b, type(b)
2 <type 'int'>

>>> s=eval(raw_input("Ingrese s :"))
Ingrese s: 2*3

>>> print s, type(s)
6 <type 'int'>

>>> m=eval(raw_input("Ingrese m :"))
```

```
>>> print b, type(b)
2 <type 'int'>

>>> s=eval(raw_input("Ingrese s :"))
Ingrese s: 2*3

>>> print s, type(s)
6 <type 'int'>

>>> m=eval(raw_input("Ingrese m :"))
Ingrese m: hola
```

```
>>> print b, type(b)
2 <type 'int'>

>>> s=eval(raw_input("Ingrese s :"))
Ingrese s: 2*3

>>> print s, type(s)
6 <type 'int'>

>>> m=eval(raw_input("Ingrese m :"))
Ingrese m: hola
```

Marca un error, por qué?

Salida de datos

La mayoría de programas requiere mostrar un resultado, en ocasiones a la pantalla y en otras, como un conjunto de datos que se enviarán a un archivo.

Para facilitar la lectura del resultado conviene elegir el respectivo formato de salida.

- `objeto1, objeto2, ...`
- %formato1, %formato2, ..., %tupla

Entero	`d`
Punto flotante	`f`
Notación científica	`e`

Formato de salida de datos

```
>>> u=6543
```

Formato de salida de datos

```
>>> u=6543

>>> v=1234.56789
```

Formato de salida de datos

```
>>> u=6543

>>> v=1234.56789

>>> print u, v
```

Formato de salida de datos

```
>>> u=6543

>>> v=1234.56789

>>> print u, v
6543 1234.56789
```

Formato de salida de datos

```
>>> u=6543

>>> v=1234.56789

>>> print u, v
6543 1234.56789

>>> print "u = %6d" % u
```

Formato de salida de datos

```
>>> u=6543

>>> v=1234.56789

>>> print u, v
6543 1234.56789

>>> print "u = %6d" % u
u=6543
```

Formato de salida de datos

```
>>> u=6543

>>> v=1234.56789

>>> print u, v
6543 1234.56789

>>> print "u = %6d" % u
u=6543

>>> print "u = %06d" % u
```

Formato de salida de datos

```
>>> u=6543

>>> v=1234.56789

>>> print u, v
6543 1234.56789

>>> print "u = %6d" % u
u=6543

>>> print "u = %06d" % u
u=006543
```

Formato de salida de datos

```
>>> print "v= %7.2f" % v
```

Formato de salida de datos

```
>>> print "v= %7.2f" % v
v=1234.57
```

Formato de salida de datos

```
>>> print "v= %7.2f" % v
v=1234.57

>>> print "v= %9.2f" % v
```

Formato de salida de datos

```
>>> print "v= %7.2f" % v
v=1234.57
```

```
>>> print "v= %9.2f" % v
v =    1234.57
```

Formato de salida de datos

```
>>> print "v= %7.2f" % v
v=1234.57

>>> print "v= %9.2f" % v
v =    1234.57

>>> print "v= %7.8f" % v
```

Formato de salida de datos

```
>>> print "v= %7.2f" % v
v=1234.57
```

```
>>> print "v= %9.2f" % v
v =    1234.57
```

```
>>> print "v= %7.8f" % v
v=1234.56789000
```

Formato de salida de datos

```
>>> print "v= %7.2f" % v
v=1234.57

>>> print "v= %9.2f" % v
v =   1234.57

>>> print "v= %7.8f" % v
v=1234.56789000

>>> print "v= %.2e" % v
```

Formato de salida de datos

```
>>> print "v= %7.2f" % v
v=1234.57
```

```
>>> print "v= %9.2f" % v
v =   1234.57
```

```
>>> print "v= %7.8f" % v
v=1234.56789000
```

```
>>> print "v= %.2e" % v
v=1.23e+03
```

Formato de salida de datos

```
>>> print "v= %7.2f" % v
v=1234.57

>>> print "v= %9.2f" % v
v =   1234.57

>>> print "v= %7.8f" % v
v=1234.56789000

>>> print "v= %.2e" % v
v=1.23e+03

>>> print "u= %6d y v=%8.4e" %(u, v)
```

Formato de salida de datos

```
>>> print "v= %7.2f" % v
v=1234.57
```

```
>>> print "v= %9.2f" % v
v =   1234.57
```

```
>>> print "v= %7.8f" % v
v=1234.56789000
```

```
>>> print "v= %.2e" % v
v=1.23e+03
```

```
>>> print "u= %6d y v=%8.4e" %(u, v)
u = 6543 y v=1.2346e+03
```

Ejercicio

A continuación se presenta el código que calcula el promedio de dos números que se proporcionan por el usuario.

Introduce el código por cada línea, presionando `Enter` para que se ejecute:

Código

```
A = eval(raw_input('Ingresa A: '))
Ingresa A: 10
B = eval(raw_input('Ingresa B: '))
Ingresa B: 20
PROM = (A+B)/2.0
print 'El promedio de %f y %f es %f' % (A,B,PROM)
```

Estructuras de control

En cualquier lenguaje de programación se incluye una serie de estructuras de control para ampliar las posibilidades de ejecución de un programa.

En Python, manejaremos las más comunes, que son relativamente sencillas de usar, cuidado siempre la sintaxis respectiva.

Condicionales

Una sentencia condicional permite evaluar si se cumple cierta condición, es decir, si su valor es `True`, se ejecuta una instrucción, en caso de que el valor de la condición no se cumpla, valor `False`, no se ejecuta la instrucción contenida y se sigue a la siguiente línea de código.

Ejemplo de condicional

```
1 a = input('Introduce el valor de a')
2 if a > 0:
3       print "a es positivo"
4       a = a + 1
5 elif a == 0:
6       print "a es 0"
7 else:
8       print "a es negativo"
```

Bucles

Un bucle es una sentencia que evalúa inicialmente una condición, en caso de que se cumple (valor `True`) se ejecuta(a) un conjunto de instrucciones, posteriormente, se revisa el valor de la condición, mientras sea verdadero, las instrucciones se ejecutan nuevamente.

Hay que considerar que se puede conocer de antemano, el número de repeticiones, hay que evitar los bucles infinitos, es decir, sentencias que no modifican el valor de la condición y por tanto, siempre se mantendrá sin salir del bucle.

```
1 nMax = 5
2 n = 1
3 a = []
4 while n < nMax:
5     a.append(1.0/n)
6     n = n + 1
7 print a
```

Sentencia for

```
1 lista = ['Hugo', 'Paco', 'Luis', 'McPato']
2 nombre = eval(raw_input('Teclea un nombre: '))
3 for i in range(len(lista)):
4     if lista[i] == nombre:
5         print nombre, ' es el numero ', i + 1, '
              en la lista'
6         break
7 else:
8     print nombre, ' no esta en la lista'
```

El uso de `else` en el bloque `for`

En el ejemplo anterior, vemos que hay una sentencia `else:` al mismo nivel de alineación del ciclo `for`, efectivamente, esta sentencia `else:` no pertenece al condicional `if`, sino al bloque `for`.

La sentencia `else:` la podemos utilizar tanto para los bucles `for` y `while`.

Control de errores

Cuando comenzamos a programar, nos podemos encontrar con mensajes de error al momento de ejecutar el programa, siendo las causas más comunes:

- errores de dedo, escribiendo incorrectamente una instrucción, sentencia, variable o constante.

Control de errores

Cuando comenzamos a programar, nos podemos encontrar con mensajes de error al momento de ejecutar el programa, siendo las causas más comunes:

- errores de dedo, escribiendo incorrectamente una instrucción, sentencia, variable o constante.
- errores al momento de introducir los datos, por ejemplo, si el valor que se debe de ingresar es 123.45, y si nosotros tecleamos 1234.5, el resultado ya se considera un error.

Control de errores

Cuando comenzamos a programar, nos podemos encontrar con mensajes de error al momento de ejecutar el programa, siendo las causas más comunes:

- errores de dedo, escribiendo incorrectamente una instrucción, sentencia, variable o constante.
- errores al momento de introducir los datos, por ejemplo, si el valor que se debe de ingresar es 123.45, y si nosotros tecleamos 1234.5, el resultado ya se considera un error.
- errores que se muestran en tiempo de ejecución, es decir, todo está bien escrito y los datos están bien introducidos, pero hay un error debido a la lógica del programa o del método utilizado, ejemplo: división entre cero.

Manejo de errores

```
c = 12.0/0.0
```

Manejo de errores

```
c = 12.0/0.0
```

```
Traceback (most recent call last):
File ''<pyshell#0>'', line 1, in ?
c = 12.0/0.0
ZeroDivisionError: float division
```

Manejo de errores

```
c = 12.0/0.0
```

```
Traceback (most recent call last):
File ''<pyshell#0>'', line 1, in ?
c = 12.0/0.0
ZeroDivisionError: float division
```

```
try:
     c = 12.0/0.0
except ZeroDivisionError:
     print 'Division entre cero'
```

Funciones

Con lo que hemos revisado sobre Python, tenemos elementos para iniciar la solución de problemas, una particular manera de agrupar un conjunto de instrucciones, es a través de funciones.

Las funciones intrínsecas de cualquier lenguaje son pocas, pero podemos extenderlas con funciones definidas por el usuario.

Estructura de una función

La estructura de una función en Python es la siguiente:

```
def nombre_funcion(parametro1, parametro2, ...):
    conjunto de instrucciones
    return valores_devueltos
```

donde parametro1, parametro2 son los parámetros. Un parámetro puede ser cualquier objeto de Python, incluyendo una función.

Los parámetros pueden darse por defecto, por lo que en la función son opcionales. Si no se utiliza la instrucción `return`, la función devuelve un objeto `null`

Ejemplo

```
1 >>> def cuadrados(a):
2         for i in range(len(a)):
3             a[i] = a[i]**2
4
5
6 >>> a = [1, 2, 3, 4]
7 >>> cuadrados(a)
8 >>> print a
```

Cálculo de la serie de Fibonacci

La sucesión fue descrita por Fibonacci como la solución a un problema de la cría de conejos: "Cierto hombre tenía una pareja de conejos juntos en un lugar cerrado y uno desea saber cuántos son creados a partir de este par en un año cuando es su naturaleza parir otro par en un simple mes, y en el segundo mes los nacidos parir también"

Cálculo de la serie de Fibonacci

La sucesión fue descrita por Fibonacci como la solución a un problema de la cría de conejos: "Cierto hombre tenía una pareja de conejos juntos en un lugar cerrado y uno desea saber cuántos son creados a partir de este par en un año cuando es su naturaleza parir otro par en un simple mes, y en el segundo mes los nacidos parir también"

cómo le hacemos?

Propuesta de código

```
1 a, b= 0,1
2 while b < 10:
3     print b
4     a, b = b, a+b
```

Propuesta de código

```
1 a, b= 0,1
2 while b < 10:
3     print b
4     a, b = b, a+b
```

```
1 a, b= 0,1
2 while b < 1000:
3     print b,
4     a, b = b, a+b
```

Módulos

Es una buena práctica almacenar las funciones en módulos. Un módulo es un archivo en donde se dejan las funciones, el nombre del módulo es el nombre del archivo.

Un módulo se carga al programa con la instrucción

```
from nombre_modulo import *
```

Python incluye un número grande de módulos que contienen funciones y métodos para varias tareas. La gran ventaja de los módulos es que están disponibles en internet y se pueden descargar, dependiendo de la tarea que se requiera atender.

Módulo `math`

Muchas funciones matemáicas no se pueden llamar directo del intérprete, pero para ello existe el módulo `math`.

Hay tres diferentes maneras en las que se puede llamar y utilizar las funciones de un módulo.

```
from math import *
```

De esta manera, se importan todas las funciones definidas en el módulo `math`, siendo quizá un gasto innecesario de recursos, pero también generar conflictos con definiciones cargadas de otros módulos.

```
from math import func1, func2,...
```

```
from math import func1, func2,...
```

```
>>> from math import log,sin
>>> print log(sin(0.5))
-0.735166686385
```

El tercer método que es el más usado en programación, es tener disponible el módulo:

```
import math
```

Las funciones en el módulo se pueden usar con el nombre del módulo como prefijo:

```
>>> import math
>>> print math.log(math.sin(0.5)) -0.735166686385
```

Contenido del módulo `math`

Podemos ver el contenido de un módulo con la instrucción:

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

['__doc__', '__name__', '__package__', 'acos',
'acosh', 'asin', 'asinh', 'atan', 'atan2',
'atanh', 'ceil', 'copysign', 'cos', 'cosh',
'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1',
'fabs', 'factorial', 'floor', 'fmod', 'frexp',
'fsum', 'gamma', 'hypot', 'isinf', 'isnan',
'ldexp', 'lgamma', 'log', 'log10', 'log1p',
'modf', 'pi', 'pow', 'radians', 'sin', 'sinh',
'sqrt', 'tan', 'tanh', 'trunc']
```