Tema 0 - Programación básica con Python III

Curso de Física Computacional

M. en C. Gustavo Contreras Mayén

Contenido

Contenido

Contenido

Contenido

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
```

```
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'
```

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'
```

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 cuánntos 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?

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 cuánntos 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
```

```
1 a, b= 0,1
2 while b < 1000:
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 log,sin
>>> print log(sin(0.5))
-0.735166686385
```

```
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
```

Podemos ver el contenido de un módulo con la instrucción:

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

frametitleManejo de archivos Hemos visto que cuando realizamos una serie de instrucciones con Python, obtenemos una respuesta en la ventana de la terminal. Conforme vayamos complicando los problemas a resolver, la información de respuesta que se nos presente en la pantalla, no tendría mucho sentido, ya que en el análisis de esa información se tendría que hacer con todo el conjunto de datos.

En todo lenguaje de programación se requiere el manejo de archivos, ya sea para escribir un conjunto de datos, o para leer los datos contenidos en un archivo y ocuparlos dentro del programa.

Ejemplo

Es atractiva la salida de la siguiente tabla?

```
1 for x in range(1,11):
2     print x, x*x, x*x*x
```

Usando un formato

```
1 for x in range(1,11):
2     print '%2d %3d %4d' %(x, x*x, x*x*x)
```

Ejemplo

Es atractiva la salida de la siguiente tabla?

```
1 for x in range(1,11):
2     print x, x*x, x*x*x
```

Usando un formato

```
1 for x in range(1,11):
2     print '%2d %3d %4d' %(x, x*x, x*x*x)
```

Escribiendo datos en un archivo

Una vez que ya hemos decidido el formato de los datos, ahora los enviaremos a un archivo.

```
1 miarchivo = open('midatos.dat','w')
2 for x in range(1,11):
3     print '%2d %3d %4d' %(x,x*x,x*x*x)
4     miarchivo.write('%2d %3d %4d \n' %(x,x*x,x*x*x))
5 miarchivo.close()
```

Revisa el archivo de datos en la carpeta en donde se guardó.

Archivos en Python

Los archivos en Python son objetos de tipo `file` creados mediante la función `open` (abrir). Esta función toma como parámetros una cadena con la ruta del archivo a abrir, que puede ser relativa o absoluta; una cadena opcional indicando el modo de acceso (si no se especifica se accede en modo lectura)

Modos de acceso

El modo de acceso puede ser cualquier combinación lógica de los siguientes modos:

- **r**: read, lectura. Abre el archivo en modo lectura. El archivo tiene que existir previamente, en caso contrario se lanzará una excepción de tipo IOError.
- **w**: write, escritura. Abre el archivo en modo escritura. Si el archivo no existe se crea. Si existe, sobreescribe el contenido.
- **a**: append, añadir. Abre el archivo en modo escritura. Se diferencia del modo 'w' en que en este caso no se sobreescribe el contenido del archivo, sino que se comienza a escribir al final del archivo.
- **b**: binary, binario.
- **+**: permite lectura y escritura simultáneas.

Lectura de archivos

Para la lectura de archivos se utilizan los métodos `read`, `readline` y `readlines`.

El método `read` devuelve una cadena con el contenido del archivo o bien el contenido de los primeros *n* bytes, si se especifica el tamaño máximo a leer.

Método `read`

```
completo = miarchivo.read()
```

Método `read(n)`

```
parte = miarchivo.read(512)
```

Lectura de archivos

Para la lectura de archivos se utilizan los métodos `read`, `readline` y `readlines`.

El método `read` devuelve una cadena con el contenido del archivo o bien el contenido de los primeros *n* bytes, si se especifica el tamaño máximo a leer.

Método `read`

```
completo = miarchivo.read()
```

Método `read(n)`

```
parte = miarchivo.read(512)
```

El método `readline` sirve para leer las líneas del archivo una por una. Es decir, cada vez que se llama a éste método, se devuelve el contenido del archivo desde el puntero hasta que se encuentra un carácter de nueva línea, incluyendo este carácter.

Método `readline()`

```
while True:
 linea = f.readline()
 if not linea: break
 print line
```

Por último, `readlines`, funciona leyendo todas las líneas del archivo y devolviendo una lista con las líneas leídas.

Para ocupar el contenido de datos de un archivo dentro de un programa con Python, se requiere manejar los arreglos de una manera dinámica, que veremos en su momento, ya que tendremos que apoyarnos en la librería `Numpy`.

Con los elementos de programación que hemos revisado, contamos con las herramientas necesarias para iniciar el curso, conforme se requieran nuevos elementos, se darán en su momento, pero igual, pueden ir avanzando de manera paralela por su cuenta.