numerical_methods

September 10, 2021

1 Python expresivo

No se busca que esta sea una introducción a la programación. Tutoriales sobre el lenguaje de programación Python se pueden encontrar en Internet fácilmente. Lo que se busca hacer en este capítulo es proveer una rápida recopilación de las características de Python.

1.1 £Por qué Python?

En esta guía se usa Python 3, con la idea de facilitar la aplicación de algunos de los temas que se discutirán. Con su sintaxis razonablemente simple se pueden escribir programas expresivamente, aunque a veces se sacrifique eficiencia. Las implementaciones tienen un objetivo pedagógico. Cuando se requiera más poder computacional, puede que se necesite cambiar a un lenguaje compilado. La mayoría del trabajo en supercomputadoras se realiza usando lenguajes como Fortran o C++. Sin embargo, construir un programa prototipo en Python puede ser invaluable para adquirir una mayor comprensión.

1.2 Calidad del código

Cuando se desarrollan programas grandes, seguir una guía sobre el formato que debe tener el código puede ser importante, pero la mayoría de los programas mostrados aquí serán cortos por lo que no será un tema importante. A veces, preguntas sobre cómo escribir y revisar programas son más importantes que el formato del código. Algunos consejos generales son los siguientes:

- **La legibilidad del código es importante** Asegúrese de que sus programas estén dirigidos a humanos, no computadoras. Esto significa evitar usar trucos "astutos". Por lo que debe usar buenos nombres de variables y escribir comentarios que agreguen valor (en lugar de repetir el código). El mayor beneficiado será usted, cuando regrese a sus programas meses después.
- **Sea cuidadoso, no rápido, cuando escriba código** Generalmente es más difícil debuggear código que escribirlo. En lugar de pasar dos minutos escribiendo un programa que no sirve y requiera dos horas para arreglar, intente pasar 10 minutos diseñando el código, siendo cuidadoso de convertir sus ideas a líneas de un programa. Tampoco hace daño usar Python de forma interactiva para probar componentes del código una a una o diferentes partes en conjunto.
- **Código sin probar es mal código** Asegúrese que su código funciona correctamente. Si tiene un ejemplo del que ya conoce la respuesta, asegúrese de que su código devuelva esa respuesta. Siga manualmente una variedad de casos (mentalmente o en papel). Esto y usando *print statements* puede ayudar mucho para asegurar que todo funcione como se debe.

- **Escriba funciones que hagan una cosa bien** En lugar de escribir muchas operaciones no relacionadas en secuencia, debe estructurar su código para que haga uso de funciones bien nombradas (y bien pensadas) que hagan una cosa bien.
- **Use librerías confiables** En la mayoría de esta guía se "reinventará la rueda", con el fin de entender cómo funcionan las cosas. Tiempo después en su trabajo, usted no siempre debe usar sus propias implementaciones de algoritmos estándar.

2 Resumen de características de Python

2.1 Lo básico

Python se puede usar de forma interactiva: cuando vea el comando `>>>`, también conocido como *chevron*. Pero también se puede usar como otros lenguajes, donde se almacena el código en un archivo.
Como otros lenguajes, en Python se usan variables, que pueden ser números enteros, complejos, etc. A diferencia de otros lenguajes, Python es dinámico en cuanto a los tipos de variable. Por ejemplo, `x = 0.5` crea una variable de punto flotante. Números como `0.5` o strings como `"Hello"` se conocen como *literales*. Si se desea imprimir en pantalla una variable, se usa la función `print(x)` que viene predefinida. Funciones adicionales están disponibles a través de la librería estándar, por ejemplo usted puede hacer un `import` de la librería `math` para hacer uso de la función `sqrt`.
Se pueden realizar operaciones aritméticas con las variables. `x**y` eleva la variable `x` a la potencia `y` ó `x//y` hace una *floor division* (hace la división y redondea el resultado hacia abajo).
Python también utiliza asignación aumentada, es decir, operaciones como `x += 1` o incluso asignaciones múltiples como `x, y = 0.5, "Hello"`.
Los comentarios son una característica importante de los lenguajes de programación: son texto ignorado por la computadora pero muy útil para los humanos que leen el código. Ese humano puede ser usted en unos meses, cuando se haya olvidado el propósito o los detalles del código que esté inspeccionando. Python permite escribir comentarios tanto de una línea como de múltiples líneas, usando `#` ó *docstrings* a través de tres comillas `"""`.

2.2 Control de flujo

Control de flujo se refiere a construcciones programables donde no todas las líneas del código son ejecutadas en orden. Un ejemplo es la declaración con `if`

[1]:
```
x = 0.5
if x != 0:
    print("x no es cero")
```

```
x no es cero
```

La sangría o *tabulación* es importante en Python: la línea después del `if` está tabulada, dando a entender que pertenece al escenario correspondiente. Similarmente, los dos puntos, `:`, al final de la línea que contiene al `if` es sintácticamente importante. Si se quiere tomar en cuenta otras posibilidades, se puede usar otro bloque tabulado empezando con un `else:` o con `elif x==0:`.
Otra construcción programable es ciclo `for`, que se usa cuando se desea repetir una cierta acción un número fijo de veces. Por ejemplo, escribiendo `for i in range(3):` se repite lo que sea que siga tres veces.

2.3 Estructuras de datos

Python utiliza entidades contenedoras, llamadas estructuras de datos.

- **Listas** Una lista es un contenedor de elementos. Puede aumentar cuando se necesite y los elementos pueden ser de diferentes tipos. Para crear una lista se usan corchetes, y los elementos se pueden separar por comas, por ejemplo `z = [3, 1+2j, -2.0]`
- **Tuplas** Las tuplas se pueden definir como listas inmutables. Son secuencias que no cambian ni crecen. Son definidas usando paréntesis en lugar de corchetes: `xs = (1,2,3)`, pero se pueden omitir los paréntesis.
- **Strings** Strings pueden verse como secuencias. Si `name = "Mary"` entonces `name[-1` es la letra 'y'. Como las tuplas, las strings son inmutables, y se pueden concatenar dos strings usando un `+`.
- **Diccionarios** Python también soporta diccionarios, que son llamados listas asociativas en computación. Su información se puede acceder usando strings o *floats* (números de punto flotante) como *llaves*. En otras palabras, diccionarios contienen pares de llave y valor. La sintaxis para crear uno utiliza braquets, con pares llave-valor separados por dos puntos. Por ejemplo, `htow = {1.41: 31.3, 1.45: 36.7, 1.48: 42.4}` es un diccionario que relaciona alturas con pesos. Para acceder a un valor, se utilizan corchetes con la llave correspondiente: `htow[1.45]`.

2.4 Funciones definidas por el usuario

Si en un programa se ejecutan muchas operaciones en secuencia, dentro de varios ciclos, su lógica se puede volver confusa. Por lo tanto, se pueden agrupar operaciones lógicamente relacionadas y crear lo que se llama funciones definidas por el usuario. Estas se refieren a líneas de código que no necesariamente se ejecutan en el orden en el que aparecen en el código.
Para introducir una función, se utiliza la palabra `def`, junto con un nombre y dos puntos al final de la línea, así como tabulaciones dentro del bloque de código que sigue. Por ejemplo, esta es una función que suma desde 1 hasta algún entero:

[2]:
```
def sumOfInts(nmax):
    val = sum(range(1, nmax + 1))
    return val
```

Esta función recibe un solo parámetro y devuelve un sólo valor. Pero podría recibir varios argumentos, o ninguno. Y similarmente, pudo haber impreso el resultado en pantalla, en lugar de devolverlo.

3 Errores

En este texto se usa la palabra *exactitud* para describir el acierto de un valor con su (posiblemente desconocido) valor verdadero. Por otro lado, la palabra *precisión* se usa para denotar cuántos dígitos se pueden usar en una operación matemática, aunque sean correctos o no. Un resultado inexacto se obtiene cuando hay un error. Esto puede pasar por una variedad de razones, de las que solo una es precisión limitada. Excluyendo el "error humano" e incertidumbre de las mediciones en los datos entrantes, hay generalmente dos tipos de errores con los que se trata en cómputo numérico: error de aproximación y error de redondeo. En detalle: