Repaso y Buenas prácticas de programación en Python

O. David Hincapie

Programación estructurada y métodos numéricos
Departamento de Ingeniería Mecánica
Universidad de Antioquia

1 Manejo de datos en Python

El manejo de datos es una habilidad que cualquier programador debe tener a la hora de realizar proyectos que involucren la programación. Durante la planeación y desarrollo de una aplicación, el programador define variables a través de estructuras de datos y los enlaza mediante sentencias y comandos.

A través de estas estructuras se crean objetos llamados datos que contienen información necesaria durante la ejecución de una aplicación. Estos objetos se almacenan en la memoria RAM ocupando un espacio determinado. El espacio que se necesita para guardar un objeto en memoria RAM depende del tipo de dato. Adicionalmente, la cantidad de espacio que se necesite en memoria RAM depende de la cantidad de variables que se creen. En algunos casos, se pueden llegar a tener códigos que permitan leer archivos grandes (del orden de GB, PB y TB) para realizar procesos más complejos en el mundo de la computación en cuestión de segundos.

Un dato puede tener tres posibles estados en un algoritmo:

1. **Creación:** Los objetos en cualquier lenguaje de programación se crean como "nombre_variable = valor". Es importante usar buenas prácticas de programación cuando se definen variables. Cuando se desee crear una o más variables, lo recomendable es se creen utilizando mismo idioma, esto incluye que no comience con números, que no tengan signos de puntuación ni cualquier otro signo y que la variable se separe por un identificador. Algunos ejemplos son:
   - number_of_iterations = 4
   - message_for_students = "Hay que ir a los talleres y a las asesorías"
   - finish_execution = False.
   - FinishExecution = False
2. **Manipulación:** Las variables se crean para interactuar con comandos y otras variables. Se puede definir la variable x = 5 y con esta variable hacer operaciones matemáticas, e incluso representaciones gráficas. La utilidad de la variable es definida por el programador, así como la interacción que tendrá esta variable con un código.
3. **Eliminación:** Para eliminar una variable, se usa el comando "del()". Dentro de los paréntesis, se indican aquellas variables que se deseen eliminar. Cuando se tengan variables que no se usen mucho en un algoritmo o cuando una variable ha cumplido su finalidad, se puede optar por borrar la variable para liberar espacio en memoria RAM.

En el siguiente código, se importa el modulo "os" para crear dos variables. Este módulo permite gestionar información del sistema operativo. En el comando usado en la linea 3 (getcwd()), se obtiene la carpeta donde esté contenido el archivo de python que contiene este código. El comando usado en la linea 4 (listdir()) retorna una lista de cadenas de caracteres con los nombres de los archivos que están en la carpeta por defecto.

Ejemplo de creación, manipulación y eliminación de variables

```
import os

path = os.getcwd()
objets_of_folder = os.listdir()
string_to_print = "La carpeta \t {} \t {} posee mas
    de 10 objetos"
if len(objets_of_folder)>0:
  print(string_to_print.format(path,'si'))
else:
  print(string_to_print.format(path,'no'))

del(path,objets_of_folder)
```

Se define un string que se imprime posteriormente según la cantidad de datos que contenga la carpeta. En este ejemplo, objects_of_folder permite decidir qué hará posteriormente el algoritmo, y string_to_print es modificada para luego ser mostrada en consola dependiendo del cambio que se hizo. Finalmente, se borran todas las variables creadas.

1.1 Tipos de datos

Los datos en la programación son una representación simbólica y digital de un atributo, estado o una variable cuantitativa y cualitativa. Existen áreas del conocimiento enfocadas a realizar estudios específicos en los datos. Algunos de estos involucran su almacenamiento, procesamiento, adquisición, minería, visualización, entre otros. Los principales tipos de datos son:

- **Simples:**
  - Entero (int)
  - Real (float)
  - Complejo (complex)
  - Carácter (str)
  - Lógico (bool)
- **Estructurados:**
  - Cadenas de caracteres (str)
  - Arreglos (array: numpy.ndarray)
  - Registros (tuple, list, dict, DataFrame)

Es importante que las sentencias que se definan e involucren tratamiento de datos tengan evalúen datos de la misma naturaleza. Existe una excepción involucrada a aquellos objetos relacionados con numpy que permiten operar arrays con datos escalares. Por ejemplo, el siguiente script define dos variables simbólicas a través del módulo scipy. Posteriormente, se usan estas variables para realizar integrales y solucionar la ecuación diferencial lineal de primer orden

$$y' + p(x)y = q(x)$$

Cuya solución se determina matemáticamente como

$$y(x) = \exp^{-1}\left(\int p(x)dx\right)\left[C + \int q(x)\cdot\exp\left(\int p(x)dx\right)dx\right]$$

```
import sympy as sm
from sympy import symbols
from sympy.parsing.sympy_parser import parse_expr

# Definicion de variables simbolicas
x,C = symbols('x C')
p = '1/(10-x)'
q = '-100/(10-x)'

# Solucion de la ED lineal de primer orden
phi = sm.exp(sm.integrate(p,x))
solution = C/phi+sm.integrate(phi*parse_expr(q),x)/phi
```

En la última linea, se debe operar $q(x)$ que es una de caracteres con una variable simbólica $\phi(x)$. Si no se tiene en cuenta que es necesario hacer una conversión de datos, la ejecución del programa saca un error debido a que no se pueden operar cadenas de caracteres con elementos simbólicos de scipy. Algo parecido ocurre cuando se intenta operar una cadena de caracteres o un caracter con un valor numérico (ej. '2'+3*4).

2 Módulos

El gran universo de Python puede ser agrupado a través de familias de módulos. Estos módulos son usados en muchas áreas del conocimiento para facilitar la interpretación y procesamiento de problemas complejos, optimizar tareas, entre otras ventajas. Estas familias tienen la finalidad de ofrecer herramientas para cumplir un objetivo específico. Por ejemplo:

- Análisis numérico: numpy, scipy
- Visualización: matplotlib, bokeh, plotly
- Modelos: tensorflow, sklearn
- Manejo de datos estructurados: pandas, dask
- Interfaz gráfica: tkinter, dash, ipywidgets
- Procesamiento digital de imágenes: cv2, pillow

2.1 Instalación

Para instalar módulos en Python, se puede hacer de tres formas:

- Desde Anaconda:

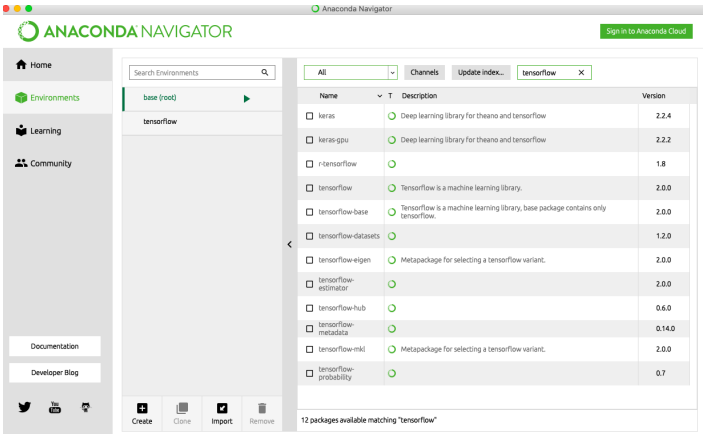

Figura 1: Ejemplo de creación, manipulación y eliminación de variables

- Desde Anaconda Prompt (Windows) ó Consola (Linux, Mac):

```
pip install modulo_1,modulo_2,...,modulo_n
```

- Desde un código:

```
!pip install modulo_1,modulo_2,...,modulo_n
```

El siguiente comando permite listar los módulos instalados en un dispositivo.

```
pip list
```

2.2 Uso

Existen varias formas de utilizar los módulos en Python, esto se define en la importación.

- Importando el módulo con el nombre que está definido por defecto. Por ejemplo:

```
import os
files = os.listdir()
```

- Importando el módulo y renombrado el nombre que está definido por defecto con un alias. Por ejemplo:

```
import numpy as np
import pandas as pd
x = np.array(10)
df = pd.DataFrame(x,columns = ['var_x'])
```

- Importando la función que se necesita del módulo. Esto es útil cuando se requieren usar una función específica de uno o varios módulos.

```
from ipywidgets import widgets, interactive
from ipywidgets import interactive_output as inter_out
from IPython.display import display

wid_slider = widgets.Slider(
            min = 0,
            max = 10,
            step =1,
            value = 1,
            description='Slider: '
            )
def update(value):
  print(value)

output = inter_out(update,{'value':wid_slider})
display(outpuy)
```

3 Documentación y buenas prácticas de programación

Cuando se escribe código, es importante tener ciertas pautas y normas para que otros programadores puedan entender y replicar nuestro trabajo. Es importante que todo el código sea escrito en un mismo lenguaje y es recomendado usar inglés. Para estas prácticas, se puede optar por usar inglés o español.

3.1 Definición de variables

Se debe definir un estándar para nombrar las variables de acuerdo a uno de los siguientes posibles casos:

- Variables definidas en minúscula separadas por underscore. (ej. str_year, int_count_cycle, df_vehicle)
- Se definen en minúscula con la primer letra de cada palabra en mayúscula. Esto se conoce comoUpperCase. (ej. StrYear, IntCountCycle, DfVehicle)

Para estas prácticas, se debe de usar el primer caso. Las variables se deben definir según la siguiente nomenclatura: **type_name** donde type representa el tipo de dato y name la descripción de la variable. Es importante ser explícitos al nombrar la variable de modo que el nombre sea coherente con el uso que tiene en el código.

3.2 Definición de funciones

Las funciones se deben nombrar con el estándar definido para las variables. Por ejemplo: si se desea crear una función que reciba un número y retorne el factorial de éste, el código con el estándar debe ser:

```
def calculate_factorial_number(int_n):
  int_result = 1
  for int_count in range(1,int_n+1):
    int_result = int_result*int_count
  return(int_result)

int_n = 3
int_fact_n = calculate_factorial_number(int_n)
print('El factorial de {} es
    {}'.format(int_n,int_fact_n)
```

3.3 Comentarios

Los comentarios permiten documentar el flujo de un código y describir funciones y algunas instrucciones claves Se hacen antes de la declaración de la sentencia a evaluar o de la función. Existen tres tipos de comentarios:

- **Por bloque:** Se usan principalmente para hacer seguimiento de funciones y métodos. Por lo general, estos comentarios van con un encabezado.

```
''' -- Declaracion funcion para calcular
    factorial --

Esta funcion calcula el factorial de un numero a
    traves de un ciclo for

- Variables entrada: int_n
- Variables de salida: int_result
'''
def calculate_factorial_number(int_n):
  int_result = 1

  for int_count in range(1,int_n+1):
    int_result = int_result*int_count
  return(int_result)

int_n = 3
int_fact_n = calculate_factorial_number(int_n)
print('El factorial de {} es
    {}'.format(int_n,int_fact_n)
```

- **Descriptivos:** Son un complemento del comentario por bloques y van en lugares donde se considere necesario dar claridad en alguna etapa del código. Estos comentarios se hacen con

```
def calculate_factorial_number(int_n):
  int_result = 1
  # Se usa el ciclo for para multiplicar
      int_result con el valor int_count que
      varia 1 a 1 hasta llevar a int_n+1
  for int_count in range(1,int_n+1):
    int_result = int_result*int_count
  return(int_result)

# Se definen variables para probar el programa
int_n = 3
# Se hace llamado de la funcion definida
    previamente
int_fact_n = calculate_factorial_number(int_n)
print('El factorial de {} es
    {}'.format(int_n,int_fact_n)
```

- **Debugging:** Son comentarios de una linea y se hace para hacer debugging y seguimiento del algoritmo.