Práctica 9: Introducción a Python

Estructuras de Datos y Algoritmos I

Autor: José Mauricio Matamoros de Maria y Campos

1. Objetivo

El alumno conocerá los fundamentos de programación en lenguage Python (versión 3.5), en particular i) variables, ii) cadenas, iii) operadores, iv) listas, v) tuplas, vi) diccionarios y vii) funciones; a fin de ser capaz de elaborar programas simples en este lenguaje de programación.

2. Material

Se asume que el alumno cuenta con un una computadora con arquitectura Intel x86 o compatible con interprete de python instalado (versión 3.5 o posterior).

2.1. Instalación de Puython 3 en Ubuntu Linux

Para instalar Python3 en Ubuntu Linux ejecute el siguiente comando en la terminal (se requieren privilegios de superusuario).

```
sudo apt-get update
sudo apt-get -y install python3-all
```

O bien, si se desea instalar sólo los paquetes mínimos necesarios

```
sudo apt-get -y install python3 python3-pip python3-numpy python3-matplotlib
```

2.2. Instalación de Puython 3 en Ubuntu Linux

Descargue e instale la última versión de Python 3 disponible en `https://www.python.org/downloads/`, por ejemplo Python 3.96 y ejecute el asistente.

Si requirere de un editor para Python se aconseja el uso de Sublime Text, Visual Studio Code o Jupyter Notebook.

3. Instrucciones

1. Lea detenidamente los antecedentes teóricos sobre los fundamentos de programación en lenguage Python 3 en la Sección 4.

2. Realice cada una de las actividades detalladas en la Sección 5 apoyándose en el código de ejemplo del Apéndice A y responda las preguntas de las mismas.

4. Antecedentes

El lenguaje Python se está volviendo cada vez más popular, y en 2017 se convirtió en el lenguaje más popular del mundo según la revista IEEE Spectrum.

¿Por qué Python es el lenuaje número uno? Porque es increíblemente fácil de aprender y usar. Parte de esto es su sintaxis simplificada y su flujo de lenguaje natural, pero mucho de esto tiene que ver con la increíble comunidad de usuarios y la variedad de aplicaciones disponibles.

Python es un lenguaje interpretado, no compilado, lo que quiere decir que el intérprete no conoce la correctez de la siguiente instrucción a ejecutar hasta que esta es proporcionada, ya sea directamente o en un archivo fuente `py`. Así, cuando se carga un archivo de código de Python, el intérprete ejecutará este línea a línea hasta el final, motivo por el cual no se necesita una función principal o `main`. Sin embargo, esto puede ocasionar que se pierda el control de la ejecución del código al trabajar en proyectos grandes o con librerías, por lo que es aconsejable siempre declarar una función principal y llamarla desde el ancla (véase Subsección 4.7).

Por ejemplo, puede intentar ejecutar en una terminal el comando `python3` y llamar a la función help.

```
$ python3
Python 3.8.5 (default)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> help()
Welcome to Python 3.8's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at https://docs.python.org/3.8/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics".  Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help>
```

o bien teclee `import this` para acceder al Zen de python

```
$ python3
Python 3.8.5 (default)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

4.1. Indentado

Un aspecto fundamental en Python es el identado. A diferencia de otros lenguajes en el que los segmentos de código y el *scope* de las variables están delimitados mediante símbolos tales como paréntesis o llaves, en python es la indentación del código lo que determina la pertenencia de una línea a un segmento de código o bloque particular y, por ende, el alcance y validez (*scope*) de las funciones.

Es por esto que las sangrías en Python no son opcionales sino que, de hecho, son necesarias y tienen un impacto considerable en cómo se ejecuta el código. Y esto viene con una ventaja: cuando se lee el código (como humano, no como computadora), es relativamente fácil ver lo que está sucediendo pues uno no se distrae con un montón de símbolos adicionales.

Es importante recalcar que cuando se escribe código en Python se prefieren cuatro espacios en lugar del caracter de tabulador. De hecho, algunos intérpretes marcarán las tabulaciones como errores.

4.2. Estructuras de Control

A diferencia de otros lenguajes de programación, Python sólo define 3 estructuras de control, cuyas condiciones no requieren de ser encerradas entre paréntesis y que siempre van sucedidas por el caracter de dos puntos (`:`). Estas estructuras de control son:

1) `if-if/else`: Evalúa la condición que aparece después de la cláusula `if` y, si esta se cumpliere, ejecuta el bloque anidado bajo el mismo. Si la condición no se cumple (es decir, se evalúa `False`), se procede con la ejecución del segmento de código anidado bajo la cláusula `else`, si existiere.

```
if a < b:
  return a
else:
  return b
```

2) `while`: Evalúa la condición que aparece después de la cláusula `while` y ejecuta el bloque anidado bajo la misma mientras ésta sea verdadera.

```
while a < b:
  a *= 2
```

3) `for`: Itera sobre una colección (tupla, lista o diccionario) y ejecuta el bloque anidado subyaciente para cada elemento.

```
for i in range(0, 100):
  print(i)
```

4.3. Variables y Tipos de Datos

Las variables son una parte esencial de Python (y de todos los lenguajes de programación). Una variable es simplemente un nombre que permite acceder a información almacenada que puede variar (cambiar).

A diferencia de otros lenguajes, las variables en Python son no tipadas, por lo que pueden almacenar cualquier tipo de información en cualquier momento, sea esta un número, una cadena, una lista, o un objeto.

Tabla 1: Tipos de datos más comúnes en Python

`bool`	Boleano (`True` y `false`)
`str`	Cadenas
`int`, `float`	Números
`list`, `tuple`,	Secuencias
`dict`,	Mapeos

En Python 3, las cadenas son arreglos de caracteres Unicode que se declaran de tres formas, entre comillas simples (`'`), entre comillas dobles (`"`) o entre tripletas de comillas simples (`'''`) para declarar cadenas que abarcan múltiples líneas. Todas las cadenas son de tipo `str` y heredan de `basestr`.

4.4. Operadores

Tabla 2: Operadores aritméticos en Python

`+`	Suma
`-`	Resta
`*`	Multiplicación
`/`	División
`%`	Módulo
`**`	Potencia
`//`	División entera

Tabla 3: Operadores de comparación en Python

`<`	Menor que
`<=`	Menor o igual que
`>`	Mayor que
`>=`	Mayor o igual que
`==`	Igual a
`!=`	Diferente de
`is`	Es
`is not`	No es

Tabla 4: Operadores lógicos en Python

`or`	O lógico
`and`	Y lógico
`not`	No lógico

Tabla 5: Otros operadores en Python

`+`	Concatenación de cadenas
`in`	Pertenencia a colección
`[]`	Acceso a elemento en colección
`.`	Pertenencia a clase o espacio

4.5. Listas, Tuplas y Diccionarios

Una lista es una colección de objetos arbitrarios, algo similar a una matriz en muchos otros lenguajes de programación, pero más flexible. Las listas se definen en Python encerrando una secuencia de objetos separados por comas entre corchetes (`[]`).

Las características más importantes de las listas son las siguientes:

- **Las listas están ordenadas:** Una lista no es simplemente una colección de objetos. Es una colección ordenada de objetos. El orden en el que especifica los elementos cuando define una lista es una característica innata de esa lista y se mantiene durante toda la vida de esa lista.

- **Las listas pueden contener cualquier objeto arbitrario:** Una lista puede contener cualquier variedad de objetos. Los elementos de una lista pueden ser todos del mismo tipo o de distintos tipos. Las listas pueden

incluso contener objetos complejos, como funciones, clases y módulos.
Es más, una lista puede contener cualquier número de objetos, desde cero hasta tantos como lo permita la memoria de su computadora. Los objetos de la lista no tienen por qué ser únicos. Un objeto determinado puede aparecer en una lista varias veces.

```
a = [1, "hola", 3.141592, False, None]
```

- **Se puede acceder a los elementos de la lista por índice.**
  Se puede acceder a los elementos individuales de una lista mediante un índice entre corchetes. Esto es exactamente análogo a acceder a caracteres individuales en una cadena. La indexación de listas se basa en cero, como ocurre con las cadenas.

```
a = [1, "hola", 3.141592, False, None]
print(a[0] + 1)
```

- **Las listas se pueden anidar con una profundidad arbitraria:** Un elemento de una lista puede ser cualquier tipo de objeto, y eso incluye otra lista. Una lista puede contener sublistas, que a su vez pueden contener las propias sublistas, y así sucesivamente con una profundidad arbitraria.

```
a = [ [None], [[None]], [[[None]]], [[[[None]]]], [[[[[None]]]]] ]
```

- **Las listas son mutables:** Una vez que se ha creado una lista, los elementos se pueden agregar, eliminar, cambiar y mover a voluntad. Python proporciona una amplia gama de formas de modificar listas.

```
a = [1, "hola", 3.141592, False, None]
a[0] = 10
```

- **Las listas son dinámicas:** Cuando se agregan elementos a una lista ésta crece según sea necesario y, de manera similar, una lista se reduce para adaptarse a la eliminación de elementos.

```
a = [1, "hola", 3.141592, False, None]
a.insert(0, 100) # a = [100, 1, "hola", 3.141592, False, None]
del a[2]         # a = [100, 1, 3.141592, False, None]
```

Python proporciona otro tipo de colección ordenada de objetos: la tupla. Las tuplas son idénticas a las listas en todos los aspectos, excepto en las siguientes propiedades:

- Las tuplas se definen encerrando los elementos entre paréntesis (`()`) en lugar de corchetes (`[]`).
- Las tuplas son inmutables, y por lo tanto...
- Las tuplas son estáticas.

Por último están los diccionarios. Los diccionarios, al igual que las listas, son objetos dinámicos y mutables que pueden contener cualquier tipo de datos y, por lo tanto, se puede anidar hasta una profundidad arbitraria. Sin embargo, a diferencia de las listas, los diccionarios no son ordenados y por lo tanto no se pueden acceder por índice. En su lugar, un diccionario asocia llaves con valores, por lo que el acceso a los elementos contenidos en un diccionario es mediante su llave. Los diccionarios se definen encerrando los elementos entre llaves (`{}`) en lugar de corchetes (`[]`), y separando las llaves de los valores con el caracter de dos puntos (`:`).

```
d = {
  1       :  100,
  "a"     :  "Hola",
  "b"     :  "Mundo",
  "llave" : None
}
d[2] = 500
print(d[1] + d[2]) # 600
```

4.6. Funciones

Una función es un bloque de código autónomo que encapsula una tarea específica o un grupo relacionado de tareas y que solo se ejecuta cuando se le llama.

En Python, una función se define usando la palabra clave `def`. Los argumentos se especifican entre paréntesis después del nombre de la función separados por comas.

```
def max(a, b):
  return a if a > b else b
```

En Python es posible asignar valores por defecto a los parámetros, colocando el valor después del nombre del argumento mediante un caracter de igual (=).

```
def pow(a, b=1):
  return a ** b
```

4.7. Ancla de programa o función `main`

Para establecer un ancla de programa o punto de arranque, es necesario indicárselo al intérprete mediante la cláusula:

```
if __name__ == '__main__':
  main()
```

Esta cláusula verificará que no exista otro punto de anclaje (por ejemplo, si el archivo se estuviera cargando como una dependencia o librería) y, en este caso, ejecutará las instrucciones que se encuentren debajo de la cláusula `if`.

Si bien puede colocarse aquí mismo la serie de instrucciones que conformarían a la función principal, por convención estas se agrupan en una función de nombre `main` como muestra el ejemplo arriba.

4.8. Estructura de un script de python

La estructura de un script de python es como sigue:

i) Linea *sheebang* que invoca al intérprete de Python
ii) Codificación del archivo (opcional)
iii) Documentación base del archivo (autor, fecha, licencia, etc.)
iv) Future imports (opcional)
v) Imports
vi) Funciones y variables (cuerpo del script)
vii) Punto de anclaje o `main` (opcional)

```python
#! /usr/bin/env python3
# -*- coding: utf-8 -*-
# ## ################################################################
# example.py
#
# Author: Mauricio Matamoros
# License: MIT
#
# ## ################################################################
import os
from math import sin, cos, pi

def main():
  pass

if __name__ == '__main__':
  main()
```

5. Desarrollo de la práctica

Lea cuidadosamente los problemas descritos a continuación y desarrolle los programas propuestos.

5.1. Actividad 1

El programa de la Apéndice A.1 calcula e imprime el seno de todos los números pasados como argumentos. Ejecútelo con la línea:

```
python3 prog1.py 1.5709
```

Modifique el programa de la Apéndice A.1 para que calcule el **coseno** de los ángulos proporcionados, ejecute el programa y responda las preguntas que se presentan a continuación.

¿Qué modificaciones se realizaron al programa? Explique [1 punto]: _____

_____

_____

¿Para qué sirve la función `float`? [1 punto] Explique: _____

_____

_____

5.2. Actividad 2

El programa de la Apéndice A.2 imprime la palabra más larga encontrada en vector de entrada. Ejecútelo con la línea:

```
python3 prog2.py Hola mundo "este es un ejemplo"
```

Modifique el programa de la Apéndice A.2 para que imprima no sólo la palabra más larga, sino también la más corta. Ejecute el programa y responda las preguntas que se presentan a continuación.

¿Qué modificaciones se realizaron al programa? Explique [1 punto]: _____

_____

¿Para qué sirve la función `len`? [1 punto] Explique:

¿Para qué sirve el método `split` de una cadena? [1 punto] Explique:

¿Para qué sirve la función `join` de una cadena? [1 punto] Explique:

5.3. Actividad 3

El programa de la Apéndice A.3 lee un archivo de texto, cuenta la cantidad de ocurrencias de cada palabra encontrada e imprime el top 10 de las palabras más usadas en ese texto. Ejecútelo con la línea:

```
python3 prog3.py
```

Modifique el programa de la Apéndice A.3 para que considere sólo palabras con **cuatro** o más caracteres y que imprima el top 3 en lugar del top 10. Ejecute el programa y responda las preguntas que se presentan a continuación.

¿Qué salida produce el programa?[1 punto]:

¿Qué modificaciones se realizaron al programa? Explique [1 punto]:

¿Se utilizan diccionarios? ¿Por qué? Explique [1 punto]:

¿Para qué sirve el método `append` de una lista? Explique [1 punto]:

A. Código de ejemplo

A.1. Archivo `src/prog1.py`

src/prog1.py

```python
import sys
from math import sin

def main(argv):
  for i in range(1, len(argv)):
    a = float(argv[i]) # The angle
    s = sin(a)         # The sine of the angle
    print("sin({}) = {}".format(a, s))

if __name__ == '__main__':
  main(sys.argv)
```

A.2. Archivo `src/prog2.py`

src/prog2.py

```python
import sys

def find_longest_word(line):
  words = line.split(" ")
  longest = words[0]
  for word in words:
    if len(word) > len(longest):
      longest = word
  return longest

def main(argv):
  if len(argv) < 2:
    return
  line = " ".join(argv[1:])
  print("Line is:", line)
  longest = find_longest_word(line)
  print("Longest is:", longest)

if __name__ == '__main__':
  main(sys.argv)
```

A.3. Archivo `src/prog3.py`

src/prog3.py

```
import os
import re

def rank_words(wc):
  ranked_words = []
  for word in wc:
    ranked_words.append( (word, wc[word]) )
  ranked_words.sort(key=lambda tup : tup[1], reverse=True)
  return ranked_words



def count_words(filepath):
  wordcount = {}
  with open(filepath, "r") as f:
    line = f.readline()
    while line:
      words = re.split(r"\W+", line)
      for word in words:
        if not word:
          continue
        word = word.lower()
        if word in wordcount:
          wordcount[word] += 1
        else:
          wordcount[word] = 1
      line = f.readline()
  return wordcount



def main():
  wordcount = count_words("quijote.txt")
  ranked = rank_words(wordcount)
  for i in range(10):
    print('Most used word #{}: "{}" ({})'
      .format(
        i+1,
        ranked[i][0],
        ranked[i][1]
      )
    )



if __name__ == '__main__':
  main()
```

B. Reporte Escrito

El reporte de la práctica deberá ser entregada en un archivo en formato `PDF` siguiendo las siguientes especificaciones:

- La primera página del documento deberá ser la carátula oficial para prácticas de laboratorio disponible en `lcp02.fi-b.unam.mx/`
- El nombre del documento `PDF` deberá ser `nn-XXXX-L09.pdf`, donde:
  - `nn` es el número de lista del alumno a dos dígitos forzosos (ej. 01, 02, etc.).
  - `XXXX` corresponden a las dos primeras letras del apellido paterno seguidas de la primera letra del apellido materno y la primera letra del nombre, en mayúsculas y evitando cacofonías; es decir, los cuatro primeros caracteres de su RFC o CURP.
- El reporte consiste en un documento de redacción propia donde el alumno explica de forma concisa y a detalle las actividades realizadas en la práctica, y reportando los resultados obtenidos.
- La longitud del reporte no deberá exceder las 3 páginas, sin contar la carátula.
- El reporte deberá seguir todos los lineamientos para documentos escritos establecidos al inicio del curso.
- Todas las referencias deberán estar debidamente citadas.

IMPORTANTE: No se aceptan archivos en otros formatos ni con nombres distintos a los especificados.