	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	M. I. Marco Antonio Martínez Quintana
Asignatura:	Estructura de Datos y Algoritmos
Grupo:	17
No de Práctica(s):	09
Integrante(s):	Díaz Segura, Mauricio Iván
No. de Equipo de cómputo empleado:	
No. de Lista o Brigada:	
Semestre:	2020-2
Fecha de entrega:	31 - III - 2020
Observaciones:	

CALIFICACIÓN: _____

OBJETIVO

Aplicar las bases del lenguaje de programación Python en el ambiente de Jupyter notebook.

INTRODUCCIÓN

Python es un lenguaje de programación de alto nivel; debido a su potencia muchas empresas e institutos lo emplean actualmente. Una de sus características es que está enfocado a la programación orientada a objetivos. Sin embargo, Python puede abarcar amplias áreas de investigación como la minería de datos o el Procesamiento del Lenguaje Natural (PLN). Asimismo, es versátil porque se puede tratar información implementada con C, el lenguaje de programación que hemos estudiado hasta ahora. Aunque, de la misma forma, Python puede emplearse como una extensión de otras aplicaciones desarrolladas. Su uso es muy amigable con el programador ya que desde el mismo código se puede consultar ayuda con respecto a las funciones de Python. De la misma forma, la filosofía de Python abarca una serie de reglas para escribir un buen código y legible.[2]

DESARROLLO Y RESULTADOS

Las variables en Python se manejan de forma diferente a C, ya que no se debe indicar el tipo de dato que tendrán almacenado. Otra particularidad es que no necesitan ser declaradas desde el principio del programa, sino en la marcha; por supuesto, cada caso tiene sus ventajas y desventajas, en este caso, la planeación del proyecto puede agregar variables que no se tenían contempladas desde el principio, pero el orden de ellas se pierde al encontrarse desperdigadas a lo largo del código.

```
x = 10      #variable de tipo entero
cadena = "Hola Mundo"     #varible de tipo cadena
print(cadena, x)
```

```
[Out] Hola Mundo 10
```

En Python, las variables son bastante prácticas porque se puede cambiar el tipo de dato que contiene en el momento en que sea necesario. Si llega a haber confusión, se puede usar `type()` para saber qué tipo de elemento contiene la variable en cuestión.

```
x = "Hola mundo!"
type(x)
```

```
[Out] str
```

Por otro lado, las cadenas son caracteres concatenados, es decir, unidos uno tras otro. Como en otros lenguajes de programación, para conocer el número de caracteres en una cadena (también sucede en listas, diccionarios y otras estructuras), Python cuenta los movimientos que se realizan, es decir, para contar el primer elemento de una cadena, no realiza movimientos y, por lo tanto, su índice será cero.

```
cadena = "Hola mundo"
cadena[0]
```

```
[Out] 'H'
```

Dos cadenas diferentes pueden juntarse en un sólo elemento.

```
cadena1 = 'Hola '
cadena2 = "Mundo"
print(cadena1)
print(cadena2)
concat_cadenas = cadena1 + cadena2 #Concatenacion de cadenas
print(concat_cadenas)
```

```
[Out] Hola

      Mundo

      Hola Mundo
```

Para combinar una salida de texto con variables hay varias maneras, entre ellas, es posible hacerlo con la función `format()`. Otra de ellas es con el signo + o, también, con comas.

```
n = 5
print("Tenemos un numero {}".format(n))
print(concat_cadenas + ' ' + str(3))
print(concat_cadenas, n)
```

```
[Out] Tenemos un numero 5

      Hola Mundo 3

      Hola Mundo 5
```

Los operadores son iguales a los de C.

```
print("5 + 4 es ", 5+4)
print(5 < 4)
```

[Out] 5 + 4 es 9

False

Las listas son estructuras que se pueden alterar y sus elementos van separados por comas. Estos últimos pueden ser de cualquier tipo, incluyendo a las listas. Se crean usando corchetes.

```
lista_numeros=[['cero', 0],['uno',1, 'UNO'], ['dos',2], ['tres', 3], ['cuatro'
    ,4], ['X',5]]
print (lista_numeros[0][1])
```

[Out] 0

Las tuplas son estructuras que no se pueden alterar. Se crean por medio de paréntesis.

```
tupla_diasDelMes=(31,28,31,30,31,30,31,31,30,31,30,31)
print (tupla_diasDelMes[5])
```

[Out] 30

De igual forma, se pueden nombrar las tuplas para poder identificarlas.

```
from collections import namedtuple
planeta = namedtuple('planeta', ['nombre', 'numero'])
planeta1 = planeta('Mercurio', 1)
print(planeta1)
```

[Out] planeta(nombre='Mercurio', numero=1)

Los diccionarios son estructuras cuyos elementos se componen de dos partes: una llave y una referencia. Para acceder a un valor o referencia, se necesita conocer la llave correspondiente, a diferencia de las listas, pues estas últimas tienen un índice. Esto significa que los diccionarios no tienen un orden puesto por la misma razón del índice. Si se necesita conocer las llaves o los valores almacenados en los diccionarios, basta con usar la función `keys()` o `items()`.

```
elementos = { 'hidrogeno': 1, 'helio': 2, 'carbon': 6 }
print (elementos['helio'])
print(elementos.keys())
```

```
[Out] 2

      dict_keys(['hidrogeno', 'helio', 'carbon'])
```

Para crear una función, es necesario escribir `def` seguido del nombre de la función. Después, entre paréntesis se colocan los parámetros con los que trabajará la función y, para continuar, se ponen dos puntos. La siguiente línea debe cumplir con la identación requerida, esto es un espacio con tabulador o cuatro con la barra espaciadora; en caso de faltar uno solo, el programa devolverá un error. Cuando se inicializa una variable dentro de la función, esa variable es de tipo local puesto que sólo funciona dentro de ella. Si se invoca la variable local en algún punto fuera de la función, el programa devolverá un error ya que no existe. Al contrario, cuando la variable se inicializa fuera de una función, es una variable global y se pueden usar en cualquier ámbito del mismo programa.

```
1 def sumar(x):
2   y = 5
3   return x + y
4 n = 4
5 sumar(n)
```

```
[Out] 9
```

Como último ejercicio, se debe obtener el área y el perímetro de tres figuras geométricas: un triangulo, un rectángulo y un trapecio. Para esto, primero se declararon las variables a usar, las cuales corresponden a las medidas de las figuras. `B` es la base mayor del trapecio, `b` es la base menor del trapecio y la base de las otras figuras, `l` son los lados del trapecio y `h` es la altura de todas las figuras. Después se definieron las funciones, las cuales consisten en hacer los cálculos por medio de los operadores aritméticos y regresarlos como dos valores. Dichos valores representan el área y el perímetro de las figuras, los cuales se almacenan en dos variables.

Finalmente, se imprime un texto que dice "El área del triángulo es" (o perímetro, según sea el caso y la figura geométrica) y se emplea la función `format()` para introducir las variables en el texto.

```
1 B = 7
2 b = 5
3 h = 8
4 l = 3
5
6 def medida_triangulo(b, h):
7   return (b*h) / 2, b*3
8
9 val1, val2 = medida_triangulo(b, h)
```

```
10
11 print("El per metro del tri ngulo es {}".format(val2))
12 print("El  rea  del tri ngulo es {}".format(val1))
13
14 def medida_rectangulo(b, h):
15   return b*h, b*4
16
17 val3, val4 = medida_rectangulo(b, h)
18
19 print("El per metro del tri ngulo es {}".format(val4))
20 print("El  rea  del tri ngulo es {}".format(val3))
21
22 def medida_trapecio(b, B, h, l):
23   return b+B+2*l, (h*(b+B))/2
24
25 val5, val6 = medida_trapecio(b, B, h, l)
26
27 print("El per metro del trapecio es {}".format(val5))
28 print("El  rea  del trapecio es {}".format(val6))
```

```
[Out] El perímetro del triangulo es 15

[Out] El área del triangulo es 20.0

[Out] El perímetro del rectangulo es 20

[Out] El área del rectangulo es 40

[Out] El perímetro del trapecio es 18

[Out] El área del trapecio es 48.0
```

CONCLUSIONES

Python es una excelente herramienta de trabajo el cual tiene una amplia gama de funciones que permite realizar tareas diversas. Aprenderlo requiere, como cualquier lenguaje de programación, de mucha constancia y paciencia.

REFERENCES

1. *Tutorial de Python.* Fecha de consulta: 31 de marzo de 2020. Recuperado de http://docs.python.org.ar/tutorial/3/real-index.html

2. *PEP 8 – Style Guide for Python Code.* Fecha de consulta: 31 de marzo de 2020. Recuperado de https://www.python.org/dev/peps/pep-0008/