



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: Marco Antonio Martinez Quintana

Asignatura: Estructura de Datos y Algoritmos I

Grupo: 17

No de Práctica(s): 9

Integrante(s): Abrego Abascal Diego

*No. de Equipo de
cómputo empleado:* -

No. de Lista o Brigada: 1

Semestre: 2

Fecha de entrega: 31/03/2020

Observaciones:

CALIFICACIÓN: _____

Introducción a Python I

Introducción

“Python es un lenguaje de programación interpretado, orientado a objetos y de alto nivel con semántica dinámica. Sus estructuras de datos integradas de alto nivel, combinadas con tipo dinámico y enlace dinámico, lo hacen muy atractivo para el desarrollo rápido de aplicaciones, así como para usarlo para scripting o pegamento para conectar componentes existentes. La sintaxis simple y fácil de aprender de Python enfatiza la legibilidad y, por lo tanto, reduce el costo del mantenimiento del programa. Python admite módulos y paquetes, lo que fomenta la modularidad del programa y la reutilización de código. El intérprete de Python y la extensa biblioteca estándar están disponibles en formato fuente o binario sin cargo para todas las plataformas principales, y se pueden distribuir libremente.

A menudo, los programadores se enamoran de Python debido a la mayor productividad que proporciona. Como no hay un paso de compilación, el ciclo de edición-prueba-depuración es increíblemente rápido. La depuración de programas de Python es fácil: un error o una entrada incorrecta nunca causará una falla de segmentación. En cambio, cuando el intérprete descubre un error, genera una excepción. Cuando el programa no detecta la excepción, el intérprete imprime un seguimiento de la pila. Un depurador de nivel fuente permite la inspección de variables locales y globales, la evaluación de expresiones arbitrarias, el establecimiento de breakpoints, el paso por el código línea por línea, etc. El depurador está escrito en Python, testimoniando el poder introspectivo de Python. Por otro lado, a menudo la forma más rápida de depurar un programa es agregar algunas declaraciones de impresión a la fuente: el ciclo rápido de edición, prueba y depuración hace que este enfoque simple sea muy efectivo.”

Traducción al español de: “*What is Python? Executive Summary*” de python.org

Objetivo

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

Desarrollo

1. Variables

```

#Iniciando variables
x = 10      #tipo entero
print(x)    #imprimir los valores de las variables

#comillas dobles o simples para crear una cadena
cadena = "Hola Mundo"    #variable de tipo cadena
print(cadena)

#Asigna un mismo valor a tres variables
x = y = z = 10
print(x,y,z)

#La función type() permite conocer el tipo de una variable
print(type(x))

print(type(cadena))

#Se pueden cambiar los valores de las variables y el tipo se cambia automáticamente
x = "Hola Mundo"
cadena = 10

print(type(x))


print(type(cadena))

SEGUNDOS_POR_DIA = 60 * 60 * 24
PI = 3.14

print(SEGUNDOS_POR_DIA, PI)

input()

```

 C:\windows\py.exe

```

10
Hola Mundo
10 10 10
<class 'int'>
<class 'str'>
<class 'str'>
<class 'int'>
86400 3.14

```

2. Cadenas

```

#Iniciando cadenas
cadena1 = 'Hola '
cadena2 = "Mundo"
print(cadena1)
print(cadena2)
concat_cadenas = cadena1 + cadena2 #Concatenación de cadenas
print(concat_cadenas)

#Para concatenar un número y una cadena se debe usar la función str()
num_cadena = concat_cadenas + ' ' + str(3) #Se agrega una cadena vacía para agregar un espacio
print(num_cadena)

#El valor de la variable se va a imprimir en el lugar donde se encuentre {} en la cadena
num_cadena = "{} {} {}".format(cadena1, cadena2, 3)
print(num_cadena)

#Cuando se agrega un número dentro de {}, el valor de la variable que se encuentra en esa posición
#dentro de la función format(), será impreso.
num_cadena = "Cambiando el orden: {1} {2} {0} {}".format(cadena1, cadena2, 3)
print(num_cadena)
input()

```

C:\windows\py.exe

```

Hola
Mundo
Hola Mundo
Hola Mundo 3
Hola Mundo 3
Cambiando el orden: Mundo 3 Hola #

```

3. Operadores

#Para el exponente se puede utilizar asterisco

```
print( 1 + 5 )
print( 6 * 3 )
print( 10 - 4 )
print( 100 / 50 )
print( 10 % 2 )
print( ((20 * 3) + (10 +1)) / 10 )
print( 2**2 )

print(False and True) #False

print (7 < 5) #Falso

print (7 > 5) #Verdadero

print ((11 * 3)+2 == 36 - 1) #Verdadero

print ((11 * 3)+2 >= 36) #Falso

print ("curso" != "CuRsO") #Verdadero

input()
```

C:\windows\py.exe

```
6
18
6
2.0
0
7.1
4
False
False
True
True
False
True
```

4. Listas

```

#Declaracion de una lista simple
lista_diasDelMes=[31,28,31,30,31,30,31,31,30,31,30,31]

print (lista_diasDelMes)      #imprimir la lista completa
print (lista_diasDelMes[0])   #imprimir elemento 1
print (lista_diasDelMes[6])    #imprimir elemento 7
print (lista_diasDelMes[11])   #imprimir elemento 12


#Declaracion de listas anidadas

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

print (lista_numeros)        #imprimir lista completa

print (lista_numeros[0])      #imprime el elemento 0 de la lista
print (lista_numeros[1])      #imprime el elemento 1 de la lista

print (lista_numeros[2][0])   #imprime el primer elemento de la lista en la posicion 2
print (lista_numeros[2][1])   #imprime el segundo elemento de la lista en la posicion 2

print (lista_numeros[1][0])
print (lista_numeros[1][1])
print (lista_numeros[1][2])

#Cambiando el valor de uno de los elementos de la lista

lista_numeros[5][0] = "cinco"
print (lista_numeros[5])

input()

```

```

C:\windows\py.exe
[31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
31
31
31
31
[['cero', 0], ['uno', 1, 'UNO'], ['dos', 2], ['tres', 3], ['cuatro', 4], ['X', 5]]
['cero', 0]
['uno', 1, 'UNO']
dos
2
uno
1
UNO
['cinco', 5]

```

5. Tuplas

```

#Declaracion de una tupla
tupla_diasDelMes=(31,28,31,30,31,30,31,31,30,31,30,31)

lista_diasDelMes=[31,28,31,30,31,30,31,31,30,31,30,31]

print (tupla_diasDelMes)      #imprimir la tupla completa
print (tupla_diasDelMes[0])   #imprimir elemento 1
print (tupla_diasDelMes[3])   #imprimir elemento 4
print (tupla_diasDelMes[1])   #imprimir elemento 2

#Declaracion de tuplas anidadas

tupla_numeros=(( 'cero', 0), ('uno',1, 'UNO'), ('dos',2), ('tres', 3), ('cuatro',4), ('X',5))


print (tupla_numeros)        #imprimir tupla completa

print (tupla_numeros[0])      #imprime el elemento 0 de la tupla
print (tupla_numeros[1])      #imprime el elemento 1 de la tupla

print (tupla_numeros[2][0])   #imprime el primer elemento de la tupla en la posición 2
print (tupla_numeros[2][1])   #imprime el segundo elemento de la tupla en la posición 2

print (tupla_numeros[1][0])
print (tupla_numeros[1][1])
print (tupla_numeros[1][2])
#Probando la mutabilidad de las listas vs la no mutabilidad de las tuplas
print("valor actual {}".format(lista_diasDelMes[0]))
lista_diasDelMes[0] = 50
print("valor cambiado {}".format(lista_diasDelMes[0]))
input()
#tupla_diasDelMes[0] = 50  #Esta asignación manda un error, ya que no se pueden cambiar los valores de las tuplas

```

 C:\windows\py.exe

```

(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
31
30
28
(( 'cero', 0), ('uno', 1, 'UNO'), ('dos', 2), ('tres', 3), ('cuatro', 4), ('X', 5))
( 'cero', 0)
( 'uno', 1, 'UNO')
dos
2
uno
1
UNO
valor actual 31
valor cambiado 50

```

6. Tupla con nombre

```

#Se debe importat la libreria para hacer uso de namedtuple
from collections import namedtuple

#Se crea la tupla con nombre
#El primer argumento es el nombre de la tupla, mientras que el segundo argumento son los campos
#p es la referencia a la tupla
planeta = namedtuple('planeta', ['nombre', 'numero'])


#Se crea el planeta 1 y se agregan a la tupla los valores correspondientes a los campos
planeta1 = planeta('Mercurio', 1)
print(planeta1)

#Se crea el planeta 2
planeta2 = planeta('Venus', 2)

#Se imprimen los valores de los campos
#Usando la referencia se llama a cada uno de sus campos
print(planeta1.nombre, planeta1.numero)
#Se obtienen los valores por el orden de los campos
print(planeta2[0], planeta2[1])

print('Campos de la tupla: {}'.format(planeta1._fields))
#Al igual que las tuplas, éstas no son mutables, si se trata de cambiar el contenido, se genera un error
#planeta1.nombre = 'Tierra'
input()

```

 C:\windows\py.exe

```

planeta(nombre='Mercurio', numero=1)
Mercurio 1
Venus 2
Campos de la tupla: ('nombre', 'numero')

```

7. Diccionarios


```

#Creando un diccionario
elementos = { 'hidrogeno': 1, 'helio': 2, 'carbon': 6 }

#El momento de la impresion, pueden aparecer en diferente orden del introducido
print (elementos)

print (elementos['hidrogeno'])

#Se pueen agregar elementos al diccionario
elementos['litio'] = 3
elementos['nitrogeno'] = 8

print (elementos) #Imprimiendo todos los elementos, nótese que los elementos no están ordenados

#Creando un nuevo diccionario
elementos2 = {}
elementos2['H'] = {'name': 'Hydrogen', 'number': 1, 'weight': 1.00794}
elementos2['He'] = {'name': 'Helium', 'number': 2, 'weight': 4.002602}

print (elementos2)

#Imprimiendo los datos de un elemento del diccionario
print (elementos2['H'])
print (elementos2['H']['name'])
print (elementos2['H']['number'])
elementos2['H']['weight'] = 4.30 #Cambiando el valor de un elemento
print (elementos2['H']['weight'])

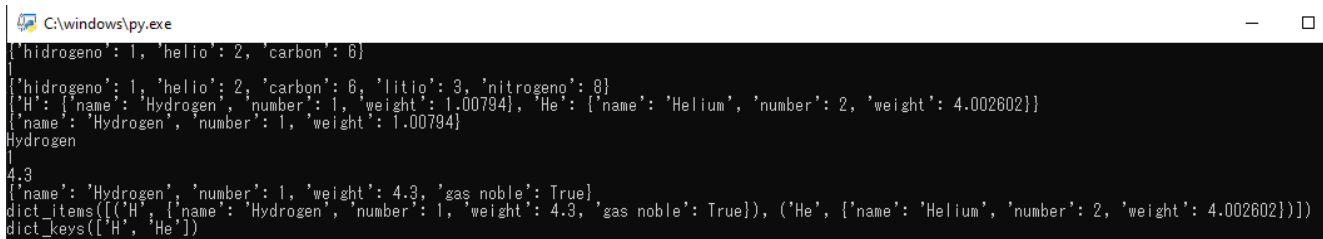
#Agregando elementos a una llave
elementos2['H'].update({'gas noble': True})
print (elementos2['H'])

#Muestra todos los elementos del diccionario
print (elementos2.items())

#Muestra todas las llaves del diccionario
print (elementos2.keys())

input()

```



```

C:\windows\py.exe
{'hidrogeno': 1, 'helio': 2, 'carbon': 6}
{'hidrogeno': 1, 'helio': 2, 'carbon': 6, 'litio': 3, 'nitrogeno': 8}
{'H': {'name': 'Hydrogen', 'number': 1, 'weight': 1.00794}, 'He': {'name': 'Helium', 'number': 2, 'weight': 4.002602}}
{'name': 'Hydrogen', 'number': 1, 'weight': 1.00794}
Hydrogen
4.3
{'name': 'Hydrogen', 'number': 1, 'weight': 4.3, 'gas noble': True}
dict_items([('H', {'name': 'Hydrogen', 'number': 1, 'weight': 4.3, 'gas noble': True}), ('He', {'name': 'Helium', 'number': 2, 'weight': 4.002602})])
dict_keys(['H', 'He'])

```

8. Funciones

```

#Las funciones pueden recibir n número de parámetros, no se necesita indicar el tipo
def imprime_nombre(nombre):
    print("hola "+nombre) #Las cadenas se pueden concatenar con el +

#Llamada a la función
imprime_nombre("JJ")

#Definiendo una función que regresa el cuadrado de un número
def cuadrado(x):
    return x**2

x = 5
#La función format() sirve para convertir los parámetros que recibe, en cadenas; éstos valores son reemplazados
#por las llaves de la cadena.
print("El cuadrado de {} es {}".format(x, cuadrado(x))) #La función cuadrado() regresa un valor

#Definiendo una función que regrese más de un valor
def varios(x):
    return x**2, x**3, x**4

#Los valores que regresa la función pueden ser guardados en variables separadas por ,
val1, val2, val3 = varios(2)
print("{} {} {}".format(val1, val2, val3))

#Función con un parámetro con un valor por defecto
def cuadrado_default(x=3):
    return x**2

#Como la función tiene un valor por default, si se manda llamar la función sin especificar el parámetro, se toma el que
#tiene por defecto
cuadrado_default()

#La función regresa tres valores, pero sólo nos interesa el primero y el tercero
val4, _, val5 = varios(2)
print("{} {}".format(val4, val5))

input()

```

C:\windows\py.exe

```

hola JJ
El cuadrado de 5 es 25
4 8 16
4 16

```

```

#Variables Globales
#Se crea una variable en el espacio global de nombres
vg = 'Global'

#Se crea una función que imprime la variable global
def funcion_v1():
    print(vg)

#Llamada a la función que imprime la variable global
funcion_v1()

#Imprime la variable global
print(vg)

#Se crea una variable local que tiene el mismo nombre que la variable global
def funcion_v2():
    vg = "Local"
    print(vg)

#Llamada a la función
funcion_v2() #Imprime valor local

#Imprime la variable global
print(vg)

#Se trata de imprimir el valor de la variable global, a diferencia de la función_v1(), se creó en el
#espacio local de la función_v2() una variable con el mismo nombre, por lo que se reemplaza la variable
#global
def funcion_v3():
    print(vg)
    vg = "Local"
    print(vg)

#Como se tiene una variable local y no se le ha asignado un valor, se genera un error
#funcion_v3()

#Para resolver el problema anterior y especificar que se quiere hacer uso de la variable global dentro de la
#función funcion_v4(), se tiene que agregar la palabra reservada global
def funcion_v4():
    global vg
    print(vg)
    vg = "Local"
    print(vg)

#Al momento de ejecutar la función se imprime el valor que tenía asignado vg antes de ser modificado por la función.
#Después de asignar el valor, éste es impreso
funcion_v4()
#Se imprime la variable global con su valor modificado

```

```

Global
Global
Local
Global
Global
Local
Local

```

10. Calculadora de Áreas

```

op=0
PI=3.1416
def menu():
    print("\nCalculadora de areas!!")
    print("\nSeleccione la figura\n")
    print("1.Triangulo\n2.Circulo\n3.Rectangulo\n4.Trapezio isoceles\n5.Salir")

def triangulo():
    b=int(input("Ingresa la base del triangulo: "))
    h=int(input("Ingresa la altura del triangulo: "))
    print("Area = "+str((b*h)/2))
    print("Perimetro = "+str(b*3))

def rectangulo():
    b=int(input("Ingresa la base del rectangulo: "))
    h=int(input("Ingresa la altura del rectangulo: "))
    print("Area = "+str(b*h))
    print("Perimetro = "+str(b*2+h*2))

def circulo():
    r=int(input("Ingresa el radio de tu circulo: "))
    print("Area = "+str((r*r)*PI))
    print("Perimetro = "+str(PI*(r*2)))

def trapezio():
    bmay=int(input("Ingresa la base mayor del trapezio: "))
    bmen=int(input("Ingresa la base menor del trapezio: "))
    h=int(input("Ingresa la altura del trapezio: "))
    trapezios=((bmay+bmen)*h)/2
    print("Area = "+str(trapezios))
    x=(bmay-bmen)/2
    l=((x*x)+(h*h))**(1/2)
    print("Perimetro = "+str(bmay+bmen+2*l))

while(op!=5):
    menu()
    op = int(input())
    if(op==1):
        triangulo()
    elif(op==2):
        circulo()
    elif(op==3):
        rectangulo()
    elif(op==4):
        trapezio()
    elif(op==5):
        print("\nHasta la proxima!!")
    else:
        print("Elige una opcion valida")

input()

```

C:\windows\py.exe

Calculadora de areas!!

Seleccione la figura

- 1.Triangulo
- 2.Circulo
- 3.Rectangulo
- 4.Trapezio isocetes
- 5.Salir

1
Ingresa la base del triangulo: 6
Ingresa la altura del triangulo: 3
Area = 9.0
Perimetro = 14.48528137423857

Calculadora de areas!!

Seleccione la figura

- 1.Triangulo
- 2.Circulo
- 3.Rectangulo
- 4.Trapezio isocetes
- 5.Salir

2
Ingresa el radio de tu circulo: 5
Area = 78.53999999999999
Perimetro = 31.416

C:\windows\py.exe

Calculadora de areas!!

Seleccione la figura

- 1.Triangulo
- 2.Circulo
- 3.Rectangulo
- 4.Trapezio isocetes
- 5.Salir

3
Ingresa la base del rectangulo: 10
Ingresa la altura del rectangulo: 5
Area = 50
Perimetro = 30

Calculadora de areas!!

Seleccione la figura

- 1.Triangulo
- 2.Circulo
- 3.Rectangulo
- 4.Trapezio isocetes
- 5.Salir

4
Ingresa la base mayor del trapezio: 6
Ingresa la base menor del trapezio: 5
Ingresa la altura del trapezio: 4
Area = 22.0
Perimetro = 19.06225774829855

Calculadora de areas!!

Seleccione la figura

- 1.Triangulo
- 2.Circulo
- 3.Rectangulo
- 4.Trapezio isocetes
- 5.Salir

6
Elige una opcion valida

Calculadora de areas!!

Seleccione la figura

- 1.Triangulo
- 2.Circulo
- 3.Rectangulo
- 4.Trapezio isocetes
- 5.Salir

5
Hasta la proxima!!

Conclusión

En esta práctica se revisaron conceptos base del lenguaje de programación Python, como son: variables, manejo de cadenas de texto, operadores y funciones entre otros. Con esto se cimentaron las bases de manera practica del lenguaje de programación y a su vez se comprobó su sencillez y eficiencia a la hora de crear programas. Dicha eficiencia es proporcionada en gran medida por las características de alto nivel que proporciona el lenguaje, como su sintaxis simplificada que como consecuencia genera que la lectura y depuración del código se haga de manera rápida y sin complicaciones innecesarias por falta de orden y claridad. También es importante mencionar la gran importancia que representa que el lenguaje al ejecutarse con errores no regrese "*segmentation fault*" y en cambio regrese el seguimiento de la pila donde se presentó el error ya que esto como consecuencia genera que el debugging también se haga de manera mucho más rápida que en lenguajes que no cuentan con esta característica.

Al final de la practica se llevo a cabo una calculadora de áreas de formas geométricas donde se terminaron de acentuar los conocimientos adquiridos en esta práctica.

Concluyo que los objetivos de la practica se cumplieron satisfactoriamente ya que el ejercicio final se pudo realizar sin complicaciones.

Bibliografía

- Apuntes de clase
- http://lcp02.fi-b.unam.mx/static/docs/PRACTICAS_EDA1/eda1_p9.pdf
- <https://www.python.org/doc/essays/blurb/>