PROGRAMACIÒN ORIENTADA A OBJETOS

Generalidades y sintaxis del lenguaje PYTHON	PYTHON

Profesor: Jaime Alberto Guzmán Luna

Generalidades de Python

Python es un lenguaje del tipo interpretado, multiparadigma: – Soporta orientación a objetos (OOP). - Programación imperativa y funcional. - Es de tipado dinámico, multiplataforma y multipropósito. Se usa principalmente para el desarrollo web y de aplicaciones informáticas.

Interpretado

los lenguajes Interpretados son aquellos en los que el código es traducido mediante un intérprete a medida que es necesario. Entre los más comunes encontramos “Python“, “Ruby”, “Javascript”, etc.
Algunas ventajas son
-Al ser interpretado no necesitamos compilar ahorrándonos tiempo en el desarrollo y testeo de una aplicación.
Nuestro código fuente puede ser ejecutado en cualquier software siempre y cuando este disponga del intérprete (Windows, Linux, Mac, Android, Web).

Python es relativamente simple, por lo que es fácil de aprender, ya que requiere una sintaxis única que se centra en la legibilidad. Los desarrolladores pueden leer y traducir el código Python mucho más fácilmente que otros lenguajes.

Por tanto, esto reduce el costo de mantenimiento y de desarrollo del programa porque permite que los equipos trabajen en colaboración sin barreras significativas de lenguaje y experimentación.

Tipado dinámico

Tipado dinámico es cuando una variable puede tomar diferentes valores de distintos tipos en diferentes momentos. En python las variables son declaradas por su contenido y no por su contenedor, lo que nos va a permitir cambiar el valor y tipo de una variable durante la ejecución sin necesidad de volver a declarar.

```
1  a = 1
2  print(type(a))
3  #type ‘int’
4  a  = “b”
5  print (type(a))
6  #type ‘str’
8  a  = []
9  print (type(a))
10 #type ‘list’
```

Básicamente vamos a guardar en ella lo que queremos y python automáticamente detectara su tipo, no hace falta que se lo indiquemos.

PROGRAMACIÒN ORIENTADA A OBJETOS

Tipos de datos

Número Entero (int)

Este tipo de dato se corresponde con números enteros, es decir, sin parte decimal.

Número Decimal (float)

Este tipo de dato se corresponde con números reales con parte decimal. Cabe destacar que el separador decimal en Python es el punto (.) y no la coma (,).

Carácter (chr)

Este tipo de dato se corresponde con un símbolo tipográfico, es decir, una letra, número, coma, espacio, signo de **puntuación**, etc.

Cadena de Texto (str)

Este tipo de datos se corresponde con una cadena de caracteres.

Booleano (bool)

Este tipo de dato reconoce solamente dos valores: Verdadero (True) y Falso (False)

Conversión de datos(Cast)

Hacer un cast o conversión significa como se dice cambiar un tipo de dato a otro, pero existen dos diferentes tipos de cast:

Conversión implícita

Esta conversión de tipos es realizada automáticamente por Python, prácticamente sin que nos demos cuenta. Sucede cuando realizamos ciertas operaciones con dos tipos distintos, pero es importante saber lo que sucede por debajo para evitar problemas.

```
1  a = 1  # int
2  b = 2.3 # float
3
4  a = a + b
5  print(type(a))  # float
6
7  # hay casos donde por obvias razones no se puede realizar
8
9  a = 1  # int
10 b = “2.3”  # string
11
12 c = a + b   # error de tipo
```

Conversión explicita

También podemos hacer conversiones entre tipos o cast de manera explícita haciendo uso de diferentes funciones que nos proporciona Python. Las más usadas son las siguientes:

float()

```
1  a = “3.5”  # string
2  a = float(a) # float
```

str()

```
1  a = 3.5  # float
2  a = str(a) # string
3
4  a = 3  # int
5  a = str(a)  # string
```

int()

```
1  a = “3”  # string
2  a = int(a) # int
3  # No es posible castear int a cualquier string
4  a = “Hola”  # string
5  a = int(a)  # Error de valor
6
7  a = 3.5  # float
8  a = int(a) # int
9  print(a) # 3
10
```

Y algunas otras como list(), set(), hex(), oct() y bin().

Imprimir por consola

En Python si deseamos imprimir por consola, lo haremos de la siguiente manera:

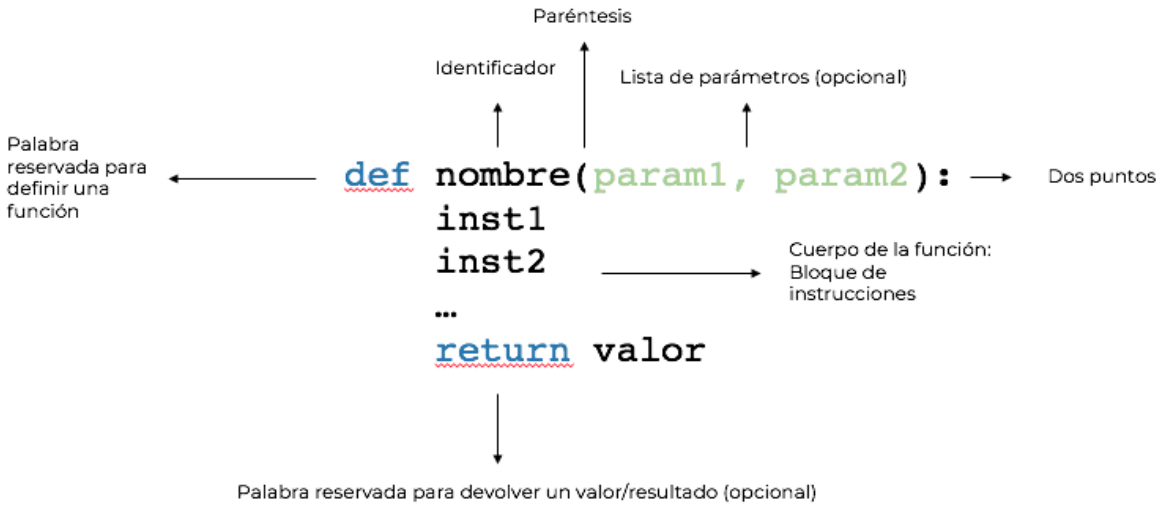
```
print("texto");
```

```
1
2  print(“Hola Mundo”)
3
```

Al correr el archivo Python, el resultado en consola será “Hola Mundo”

Funciones

Como definir una función en Python



PROGRAMACIÒN ORIENTADA A OBJETOS

Ejemplo 1

```
1  from datetime import datetime
2
3
4  def bienvenida():
5       now = datetime. Now
6       return “Bienvenido al servidor, entraste a " + str(now)”
7
8  print(bienvenida())
```

Ejemplo 2

```
1
2  def suma(a, b):
3      s = a + b
4      return s
5
6  print(suma(100,100))
```

Ahora veamos algunas funciones específicas de Python.

max()

Esta función devuelve el elemento más grande en una lista o el más grande de dos o más argumentos.

```
1  print(max(2,3))
2  # imprime 3
3  print(max([1,2,3,4])
4  # imprime 4
5  print(max(“a”,”b”,”c”))
6  # imprime c
7  print(max(2,”a”))
8  # argumentos invalidos
```

min()

Esta función devuelve el elemento más pequeño en un iterable o el más pequeño de dos o más argumentos.

```
1  print(min(2,3))
2  # imprime 2
3  print(min ([1,2,3,4])
4  # imprime 1
5  print(min(“a”,”b”,”c”))
6  # imprime a
7  print(min(2,” a”))
8  # argumentos inválidos
```

len()

Esta función devuelve la longitud (el número de elementos) de un objeto. Toma un argumento que puede ser una secuencia (como un String) o una colección.

```
1  list = [1,2,3,4,5]
2  print(len(list))
3  # imprime 5
4  print(len("Hola Mundo")
5  # imprime 10
```

input()

Esta función permite obtener texto escrito por teclado. Al llegar a la función, el programa se detiene esperando que se escriba algo y se pulse la tecla Intro.

```
1  print("¿Cómo se llama?")
2  nombre = input()
3  print(f"Me alegro de conocerle, {nombre}")
4
5  nombre = input("¿Cómo se llama? ")
6  print(f"Me alegro de conocerle, {nombre}")
```

Operadores de comparación

Los operadores de comparación se utilizan para comparar dos o mas valores. Y el resultado de este siempre es True o False.

> Mayor que
>= Mayor o igual que
< Menor que
<= Menor o igual que
== Igual
!= Distinto

```
1  import random
2
3  a = random.randint(0,100)
4  b = random.randint(0,100)
5  print("a = "+str(a))
6  print("b = "+str(b))
7  print("a > b es " + str(a > b))
8
```

PROGRAMACIÒN ORIENTADA A OBJETOS

Estructuras de datos

Listas

Las listas o arrays en Python son estructuras de datos muy flexibles en las que podemos mezclar valores de varios tipos, o solo de un tipo. Su declaración es sencilla y obedece un formato JSON estándar.

```
1  a = [1,3,4,5]
2  b = [1.4,3,”a”]
3  c = [4,1,[3,2]]  #lista ortogonal
4
5  #append y extend
6  a.append(“c”)
7  # agrega el elemento c al final de la lista a=[1,3,4,5,”c”]
8  b.extend([“b”,”c”,”d”])
9  #agrega la lista al final de la lista b=[1.4,3,”a”,”b”,”c”,”d”]
10
11 #index: nos devuelve la posición del elemento indicado
12 a.index(3)  # 1
```

```
1  a = [1,3,4,5]
2  b = [1.4,3,”a”]
3  c = [4,2,3,2]
4
5  # insert: inserta un elemento en la lista según el índice puesto
6  a.insert(2,7)  # a=[1,3,7,4,5]
7
8  #remove: remueve un elemento de la lista que desees(a diferencia del
9  pop() que devuelve el ultimo elemento de la lista y lo remueve de esta)
10 b.remove(3)  # b=[1.4,”a”]
11
12 #count: devuelve el número de elementos que hay actualmente en la lista
13 a.count(1)  # 1
14 c.count(2)  # 2
```

PROGRAMACIÒN ORIENTADA A OBJETOS

Tuplas

Las tuplas son estructuras de datos parecidas a las listas, excepto que en este caso son inmutables. Una tupla consiste en valores separados por comas. Algunas de las funciones de las listas también funcionan en las tuplas como len y la posibilidad de obtener porciones del arreglo.

```
1  a = (1,3,4,5)
2
3  len(a)    # 4
4
5  a[1:]     # (3,4,5)
6
7  #Algo importante en las tuplas es que como los paréntesis se usan en
8  Python para agrupar expresiones, si quisiéramos crear una tupla con un
9  solo valor simplemente agregamos una coma al final
10
11 b = (1,)
12
13
14
```

Sets

Los sets en Python son listas sin un orden específico pero cuyos elementos son únicos, es decir, no existe la repetición. Estos se pueden definir explícitamente con la función set o también podemos utilizar llaves

```
1  a = set([1,2,3,1])  #  {1,2,3}
2  b = {1,2,3,1}       #  {1,2,3}
3
4  #podemos agregar nuevos elementos con la función add
5  a.add(4)     #  {1,2,3,4}
6  #podemos revisar si hay un elemento en el set usando la función in
7  1 in a       #  True
```

Diccionarios

Un diccionario en Python actúa de manera similar a una lista, excepto que el índice de este no necesariamente tiene que ser un numero entero, y se asemejan bastante a un objeto JSON

```
1  a = {“x”:1,”y”:2,”z”:3}
2  a[“x”]    #  1
3
4  #Para borrar un elemento de un diccionario, utilizamos la función del
5  del(a[“x”])   #  {“y”:2,”z”:3}
6
7  #A continuación veremos algunas funciones para devolver los valores de
8  las llaves y los ítems
9
10 a.keys()    #  [“x”,”y”,”z”]
11 a.values()  #  [1,2,3]
12 a.items()   #  [(“x”,1),(“y”,2),(“z”,3)]
13
14
```

Estructuras condicionales
IF

if (condición):
código

La condición evalúa a verdadero o falso, el código se ejecuta si la condición es verdadera

```
1  x = 3
2  if(x == 3):
3        print(“x es igual a 3”)
```

ELSE IF

if (condición):
códigoT
else:
codigoF

La condición evalúa a verdadero o falso, el códigoT se ejecuta si la condición es verdadera y existe un bloque opcional de códigoF si la condicion es falsa

```
1  x = 0
2  if(x == 3):
3        print(“x es igual a 3”)
4  else:
5        print(“x es diferente de 3”)
6
7
```

ELIF

if(condición):
    CodigoT
elif(condición):
    CodigoE
else:
    CodigoT

En python no existen switches, como en Java. Pero podemos usar la sentencia elif que seria como un else if y asi podemos “imitar” a un switch.

```
1  dia="Sabado"
2
3  if(dia == "Lunes"):
4      print("Lunes otra vez no!!")
5  elif(dia == "Martes"):
6      print("Los martes son pesados")
7  elif(dia == "Miercoles"):
8      print("Casi mitad de semana")
9  elif(dia == "Jueves"):
10     print("Jueves como me gustas")
11 elif(dia == "Viernes"):
12     print("Por fin Viernes :)")
13 elif(dia == "Sabado"):
14     print("Sabadoooo")
15 elif(dia == "Domingo"):
16     print("Mañana sera LUNESSS")
17 else:
18     print("Eso no es un dia")
```

Estructuras cíclicas(Loops)

Son estructuras repetitivas, que se ejecutan iterativamente hasta cumplir o dejar de cumplir una condición

Bucle While

Este bucle, se encarga de ejecutar una misma acción "mientras que" una determinada condición se cumpla.

```
1  año = 2001
2  while año <= 2007:
3        print(“informes del año” + str(año))
4        año += 1
5  #informe del año 2001
6  #informe del año 2002
7  #informe del año 2003
8  #informe del año 2004
9  #informe del año 2005
10 #informe del año 2006
11 #informe del año 2007
12
```

Bucle For

El bucle for, en Python, es aquel que nos permitirá iterar sobre una variable compleja, del tipo lista o tupla.

```
1  #Por cada nombre en mi lista imprimirlo.
2  nombres = [“Juan”,”Pedro”,”Luis”,”Miguel”]
3  for nombre in nombres:
4        print(nombre)
5
6  # Juan
7  #Pedro
8  #Luis
9  #Miguel
10
```

```
1  #Por cada fruta en mi tupla imprimirla.
2  frutas = (“manzana”,”mango”,”banano”,”uva”]
3  for fruta in frutas:
4        print(“Comiendo “ + fruta)
5
6  # Comiendo manzana
7  #Comiendo mango
8  #Comiendo banano
9  #Comiendo uva
10
```