

Módulo 1 – INTRODUCCIÓN A PYTHON





WHY PYTHON?



PYTHON

- Python fue creado por Guido van Rossum (<http://www.python.org/~guido/>)
 - Da este nombre al lenguaje inspirado por el popular grupo cómico británico Monty Python
- Guido creó Python durante unas vacaciones de navidad en las que (al parecer) se estaba aburriendo

CARACTERÍSTICAS DE PYTHON I

- Muy legible y elegante
 - Imposible escribir código ofuscado
- Simple y poderoso
 - Minimalista: todo aquello innecesario no hay que escribirlo (;, {},), '\n')
 - Muy denso: poco código hace mucho
 - Soporta objetos y estructuras de datos de alto nivel: strings, listas, diccionarios, etc.
 - Múltiples niveles de organizar código: funciones, clases, módulos, y paquetes
 - Python standard library (<http://www.python.org/doc/current/lib/lib.html>) contiene un sinfín de clases de utilidad
 - Si hay áreas que son lentas se pueden reemplazar por plugins en C o C++, siguiendo la API para extender o empotrar Python en una aplicación, o a través de herramientas como SWIG, sip o Pyrex.

CARACTERÍSTICAS DE PYTHON II

- De scripting
 - No tienes que declarar constantes y variables antes de utilizarlas
 - No requiere paso de compilación/linkage
 - La primera vez que se ejecuta un script de Python se compila y genera bytecode que es luego interpretado
 - Alta velocidad de desarrollo y buen rendimiento
- Código interoperable (como en Java "write once run everywhere")
 - Se puede utilizar en múltiples plataforma (más aún que Java)
 - Puedes incluso ejecutar Python dentro de una JVM (Jython)
- Open source
 - Razón por la cual la Python Library sigue creciendo
- De propósito general
 - Puedes hacer en Python todo lo que puedes hacer con C# o Java, o más

Peculiaridades sintácticas

- Python usa tabulación (o espaciado) para mostrar estructura de bloques
 - Tabula una vez para indicar comienzo de bloque
 - Des-tabula para indicar el final del bloque

Código en C/Java	Código en Python
<pre>if (x) { if (y) { f1(); } f2(); }</pre>	<pre>if x: if y: f1() f2()</pre>

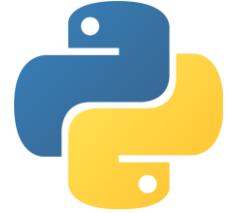
PYTHON VS. JAVA

- Java es un lenguaje de programación muy completo que ofrece:
 - Amplio abanico de tipos de datos
 - Soporte para threads
 - Strong typing
 - Y mucho más ...
- Python es un lenguaje de scripting:
 - No ofrece strong typing
 - Bueno para prototipos pero malo para grandes sistemas
 - Puede cascar en tiempo de ejecución
 - Todo lo que puedes hacer con Java también lo puedes hacer con Python
 - Incluso puedes acceder a través de Python a las API de Java si usas Jython (<http://www.jython.org>)

¿Para qué [no] es útil?

- Python no es el lenguaje perfecto, no es bueno para:
 - Programación de bajo nivel (system-programming), como programación de drivers y kernels
 - Python es de demasiado alto nivel, no hay control directo sobre memoria y otras tareas de bajo nivel
 - Aplicaciones que requieren alta capacidad de computo
 - No hay nada mejor para este tipo de aplicaciones que el viejo C
- Python es ideal:
 - Como lenguaje "pegamento" para combinar varios componentes juntos
 - Para llevar a cabo prototipos de sistema
 - **Para la elaboración de aplicaciones cliente**
 - **Para desarrollo web y de sistemas distribuidos**
 - **Para el desarrollo** de tareas científicas, en los que hay que simular y prototipar rápidamente

INSTALACIÓN DE PYTHON



- Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.
- Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma.



INSTALACIÓN DE PYTHON

- Desde la pagina <https://www.python.org/downloads/> podremos descargar la ultima versión disponible de Python.



python™

Donate Search GO Socialize

About Downloads Documentation Community Success Stories News Events

INSTALACIÓN DE PYTHON

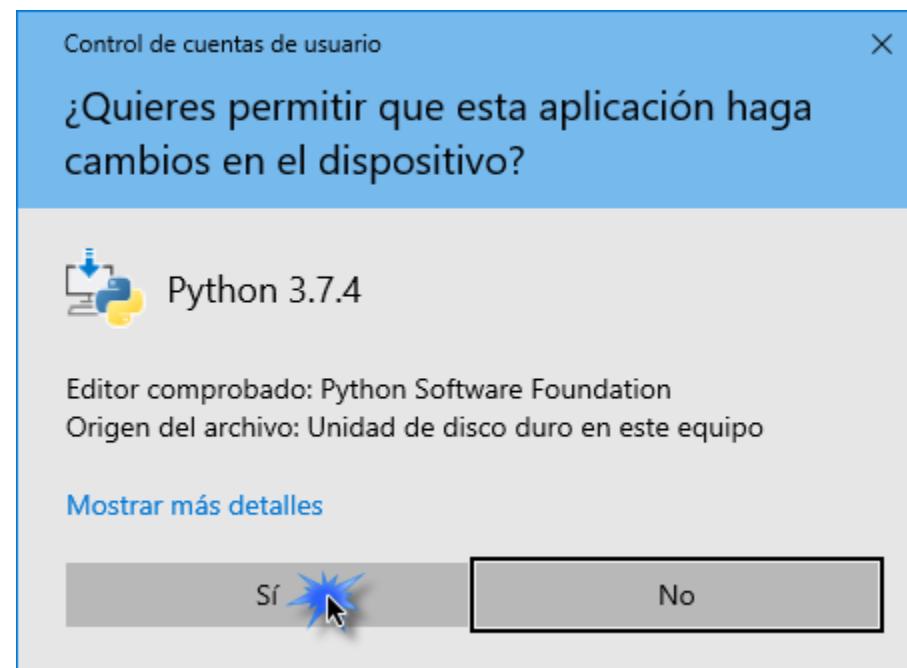


 python-3.7.4

11/10/2019 15:34

25.063 KB

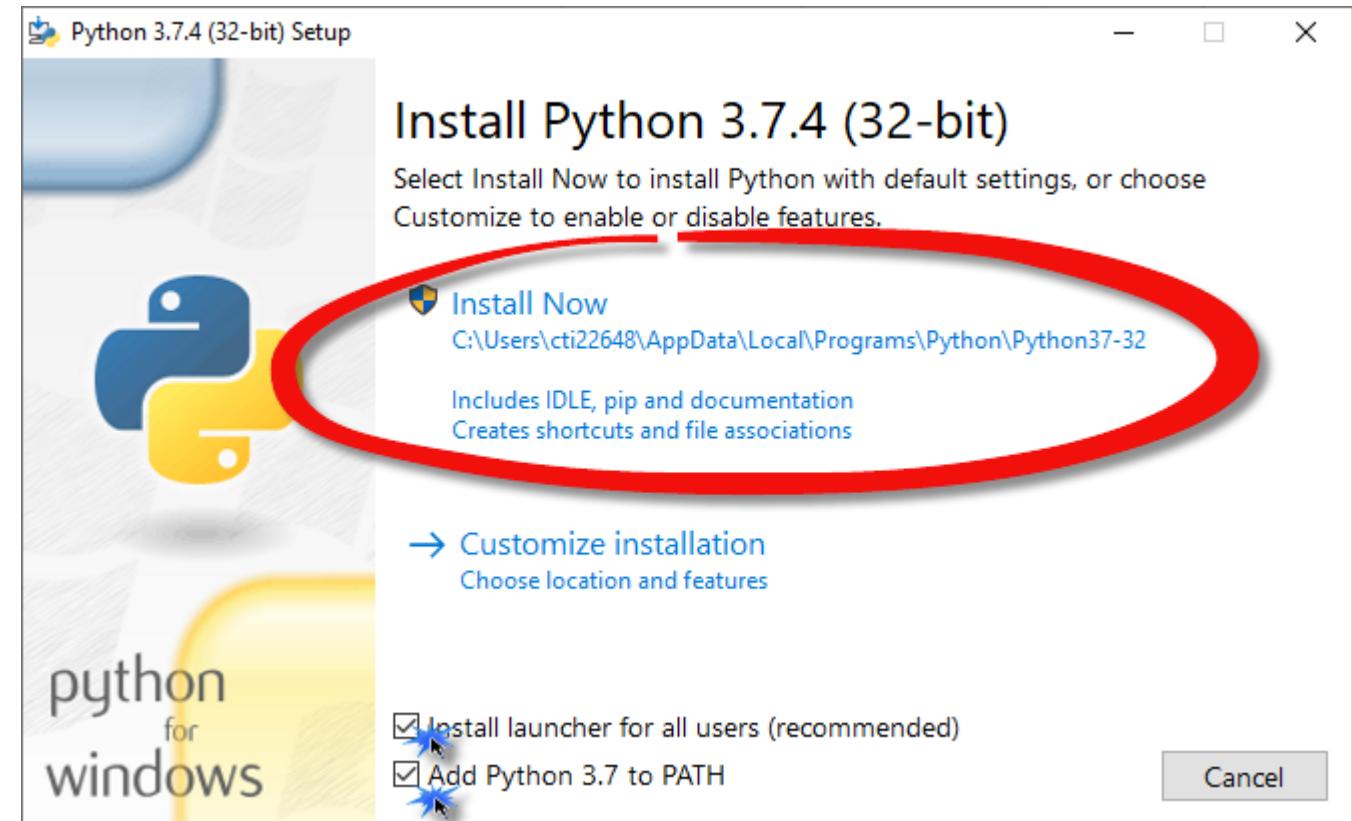
- Doble Click sobre el instalador
 - Seguimos las instrucciones de instalación



INSTALACIÓN DE PYTHON



- Es muy importante tildar la Opción: add Python to path.



INSTALACIÓN DE LIBRERIAS



- En informática, una biblioteca o llamada por vicio del lenguaje **librería** (del inglés library) es un conjunto de implementaciones funcionales, codificadas en un lenguaje de **programación**, que ofrece una interfaz bien definida para la funcionalidad que se invoca.
- PIP Install
- Pymysql
- TextFSM
- Multiping
- Netmiko

INSTALACIÓN DE LIBRERIAS



El Python Package Index o PyPI es el repositorio de software oficial para aplicaciones de terceros en el lenguaje de programación Python. Los desarrolladores de Python pretenden que sea un catálogo exhaustivo de todos los paquetes de Python escritos en código abierto

El comando que se utiliza para agregar librerías a nuestro repositorio es:

PIP Install

A screenshot of a Windows command prompt window titled "Símbolo del sistema". The window shows the following text:

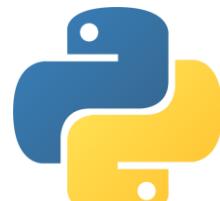
```
Microsoft Windows [Versión 10.0.17763.805]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Daniel>pip install pymysql
```

The window has standard window controls (minimize, maximize, close) at the top right.

- Para instalar librerías dentro de la red de claro utilizar el proxy
Pip install --proxy <http://usuario:password@corp-ctimovil.net:8083> librería

INSTALACIÓN DE LIBRERIAS



Símbolo del sistema

```
Microsoft Windows [Versión 10.0.17763.805]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.
```

```
C:\Users\Daniel>pip install netmiko
```

Símbolo del sistema

```
Microsoft Windows [Versión 10.0.17763.805]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.
```

```
C:\Users\Daniel>pip install textfsm
```

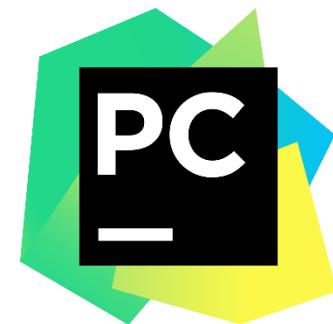
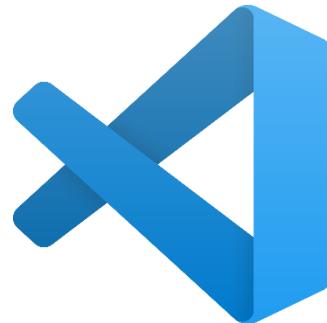
Símbolo del sistema

```
Microsoft Windows [Versión 10.0.17763.805]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.
```

```
C:\Users\Daniel>pip install multiping
```

INSTALACIÓN DE IDE

IDE: Es un entorno de desarrollo integrado o entorno de desarrollo interactivo, en inglés Integrated Development Environment (IDE), es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.





PRIMEROS PASOS

TATA TATA TATA TATA TATA TATA TATA
TATA TATA TATA TATA TATA TATA TATA

TATA CONSULTANCY SERVICES

```
>>>a = 3  
>>>a = 'Entrada'  
>>>a = 0,7172
```

```
>>> a=3  
>>> type(a)  
<class 'int'>  
>>> a = "Entrada"  
>>> type(a)  
<class 'str'>
```

```
>>> Puerto_salida = 20  
>>> id(Puerto_Salida)  
264489392
```

PYTHON – VARIABLES

La Variable no tiene un tipo, ya que el dato esta asociado a un objeto.

“a” hace referencia a diferentes objetos.

Una variable hace referencia a un determinado objeto en un determinado espacio de tiempo.

PYTHON – TIPOS DE VARIABLES

- Text Type: *str*
 - ✓ Nombre = "Maria"
 - ✓ Ip = '10.2.48.19'
- Numeric Types: *int, float, complex*
 - ✓ Edad = 20
 - ✓ Promedio= 8,92
 - ✓ x= 8j
- Sequence Types: *list, tuple, range*
 - ✓ Frutas = ["Manzana", "banana", "pera"]
 - ✓ Frutas = ("Manzana", "banana", "pera")
 - ✓ x = range(6)

PYTHON – TIPOS DE VARIABLES

- Mapping Type: dict
 - ✓ `x = {"nombre": "José", "Edad": 36}`
- Boolean Type: bool
 - ✓ `x = True`
 - ✓ `y = False`

Python – Operadores Relacionables

- Operador == (Igual)

El operador == evalúa que los valores sean iguales para varios tipos de datos.

```
>>> 5 == 3
False
>>> 5 == 5
True
>>> "Puerto" == 5
False
>>> "192.168.0.1" == " 192.168.0.1"
True
>>> type("192.168.0.1" ) == str
True
```

- Operador != (Distinto)
 - Operador < (Menor)
 - Operador > (Mayor)
 - Operador <= (Menor Igual)
 - Operador >=(Mayor Igual)

Python – Cadenas de Datos

Las cadenas no son más que caracteres encerrado entre comillas simples ('cadena') o dobles ("cadena").

<i>Tipo</i>	<i>Clase</i>	<i>Notas</i>	<i>Ejemplo</i>
<code>str</code>	<i>Cadena</i>	<i>Inmutable</i>	<code>'Hola Mundo'</code>
<code>unicode</code>	<i>Cadena</i>	<i>Versión Unicode de str</i>	<code>u'Hola Mundo'</code>

```
>>>texto1= "Hola Mundo"  
>>>texto2='Hola Mundo'  
>>>Print(texto1)  
>>>Print (texto2)  
>>>Print("Hola Mundo")
```

```
>>>texto1= "Hola "  
>>>texto2=' Mundo'  
>>>Print(texto1 + texto2)
```

Concatenación

```
>>>texto1= "Hola "  
>>>Print(texto1 *3)
```

Repetición

Python – Cadenas de Datos

```
>>>texto1= "Curso"  
>>>texto2= "Python"  
  
>>>print("Nuevo " + texto1 + " de " + texto2)
```

```
>>>texto1= "Curso"
>>>texto2= "Python"
>>>texto3="Nuevo {} de {}".format (texto1, texto2)

>>>print(texto3)
```

Python – Cadenas de Datos

Ejemplo: Escribir un programa que devuelva el nombre de la PC y su respectiva IP.

```
>>>import socket  
  
>>>host_name = socket.gethostname()  
>>>print ("Host name: %s" %host_name)
```

Host name: Eduardo-PC

```
>>>print ("IP address: %s" %socket.gethostbyname(host_name))
```

IP address: 192.168.0.77

Python – Arreglos de String

C	U	R	S	O		D	E		P	Y	T	H	O	N
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
>>>texto="Curso de Python"  
>>>letra=texto[0]  
>>>print("El valor es"+ letra)
```

La letra es C

```
>>>texto="Curso de Python"  
>>>letra=texto[-5]  
>>>print("El valor es"+ letra)
```

La letra es y

```
>>>texto="Curso de Python"  
>>>letra=texto[0:5]  
>>>print("El texto es"+ letra)
```

La letra es Curso

```
>>>texto="Curso de Python"  
>>>letra=texto[0:14:2]  
>>>print("La Salida es"+ letra)
```

La salida es Crod yh

Python – Condicional if

La sentencia condicional if se usa para tomar decisiones, este evalúa básicamente una operación lógica, es decir una expresión que de como resultado True o False, y ejecuta la pieza de código siguiente siempre y cuando el resultado sea verdadero.

A continuación un ejemplo de estructura condicional if/elif/else completo:

```
numero = int(input("\nIngresa un número entero, por favor: "))
if numero < 0:
    numero = 0
    print ('El número ingresado es negativo cambiado a cero.\n')
elif numero == 0:
    print ('El número ingresado es 0.\n')

elif numero == 1:
    print ('El número ingresado es 1.\n')
else:
    print ('El número ingresado es mayor que uno.\n')
```

Python – Condicional if

Escribir un programa que almacene la cadena de caracteres *contraseña* en una variable, pregunte al usuario por la contraseña e imprima por pantalla si la contraseña introducida por el usuario coincide con la guardada en la variable sin tener en cuenta mayúsculas y minúsculas.

Python – Condicional if

Escribir un programa que almacene la cadena de caracteres contraseña en una variable, pregunte al usuario por la contraseña e imprima por pantalla si la contraseña introducida por el usuario coincide con la guardada en la variable sin tener en cuenta mayúsculas y minúsculas.

```
key = "contraseña"
password = input("Introduce la contraseña: ")
if key == password.lower(): #Variable.lower()  minúscula ---Variable.upper() MAYUSCULA
    print("Contraseña correcta.")
else:
    print("Contraseña errónea.")
```

Python – Bucle FOR

La sentencia **for** en Python difiere un poco de lo que uno puede estar acostumbrado en lenguajes como C o Pascal. En lugar de siempre iterar sobre una progresión aritmética de números (como en Pascal) o darle al usuario la posibilidad de definir tanto el paso de la iteración como la condición de fin (como en C), la sentencia *for* de Python itera sobre los ítems de cualquier secuencia (una lista o una cadenas de caracteres), en el orden que aparecen en la secuencia.

```
animales = ['gato', 'perro', 'serpiente']
for animal in animales:
    print ("El animal es: {}".format(animal))

for animal in animales:
    cantidad_letra=len(animal)
    print ("El animal es: {}, tamaño de palabra es: {}".format(animal, cantidad_letra))
```

Python – Bucle FOR

```
IP= ['10.2.168.10', '10.95.88.78',".....", '10.5.8.18']
for i in range (0 ,len(IP)):

    print ("La ip es: {}".format(IP [ i ] ))
```

```
La ip es: 10.2.168.10
La ip es: 10.95.88.78
La ip es: ....
La ip es: 10.5.8.18
```

Python – Bucle FOR

Escriba un programa que pida un número entero mayor que cero y calcule su factorial.

- El factorial de un entero (que se escribe con una exclamación, factorial de 5 se escribe $5!$) es el producto de los enteros hasta dicho número. Es decir $5! = 5 * 4 * 3 * 2 * 1 = 120$, o lo que es lo mismo, $1 * 2 * 3 * 4 * 5$.
- Por definición, el factorial de 0 es 1, es decir $0! = 1$, por lo que el programa podría admitir también el 0.

Python – Bucle FOR

Escriba un programa que pida un número entero mayor que cero y calcule su factorial.

```
numero = int(input("INGRESE UN NUMERO\n"))
print("El numero ingresado es : ", numero)

if numero == 0:
    factorial=1
    print("El factorial de {} es = {}".format(numero, factorial))
else:
    factorial=1
    for num in range(1,numero+1):
        factorial= factorial * num

print("El factorial de {} es = {}".format(numero, factorial))
```

Python - Funciones

Una función es un bloque de código con un nombre asociado, que recibe cero o más argumentos como entrada, sigue una secuencia de sentencias, la cuales ejecuta una operación deseada y devuelve un valor y/o realiza una tarea.

El uso de funciones es un componente muy importante del paradigma de la programación llamada *estructurada*, y tiene varias ventajas:

- **modularización:** permite segmentar un programa complejo en una serie de partes o módulos más simples, facilitando así la programación y el depurado.
- **reutilización:** permite reutilizar una misma función en distintos programas.

```
import math  
decibelios= math.log(17)  
print(decibelios)
```

Python - Funciones

Son bloques que puede contener código fuente y ser invocados cuando se necesite, a continuación un ejemplo:

```
def prueba():
    """ ejemplo simple de una función """
    print ("función de prueba") ...
```

prueba()

La palabra reservada `def` se usa para definir funciones. Debe seguirle el nombre de la función en el ejemplo anterior `prueba()` y la lista de parámetros formales entre paréntesis. Las sentencias que forman el cuerpo de la función empiezan en la línea siguiente, y deben estar indentado.

Python - Funciones

```
print("Ejemplo de funciones")
```

```
def nueva_linea():
```

```
print("")
```

```
def dos_lineas():
```

nueva linea()

nueva linea()

```
print("primera linea")
```

nueva linea()

```
print("segunda linea")
```

dos lineas()

```
print("Tercera linea")
```

Python – Funciones con argumentos

Sintaxis:

```
Def nombre_funcion( valor 1, valor2, ....., valor n)
    #operaciones dentro de la función
    print("Puedo imprimir el resultado o no")
```

Valor1=xx

Valor2=xx

..

..

Valorn=xx

nombre_funcion(valor 1, valor2,, valor n)

```
def suma(a,b):
    c=a+b
    print("Suma=",c)
```

```
numero_uno=eval(input())
numero_dos=eval(input())
suma(numero_uno,numero_dos)
```

Python - Funciones

```
import ping

respuesta=ping.ping("www.google.com.ar")
if respuesta != -1:
    print("El equipo Responde")
else:
    print("El equipo NO Responde")
```



Ping.py

Python – Manejos de Archivos

Apertura de un archivo con la función *open()*:

La función *open()* tiene por objeto interactuar con el sistema de archivos local para crear, sobreescribir, leer o desplazarse dentro de un archivo ya sea de texto o binario.

<nombre> = open(<ruta del archivo>, <modo>)

Modos de abrir un archivo:

Por el tipo de archivo.

't' se trata de un archivo de texto.

'b' permite escritura en modo binario

'U' define saltos de línea universales para el modo de lectura.

Los archivos de texto y los archivos binarios representan tipos distintos en Python.

Por el tipo de acceso.

'r' es el modo de lectura.

'w' es un modo de escritura. En caso de existir un archivo, éste es sobrescrito.

'a' es un modo de escritura. En caso de existir un archivo, comienza a escribir al final de éste.

'+' es un modo de escritura/lectura.

Python – Manejos de Archivos

```
import os

NOMBRE_ARCHIVO = 'C:/...../EJEMPLO.txt'
print ("\n\nLeer un archivo")
print ("=====\\n")

f = open(NOMBRE_ARCHIVO, 'r')
s = f.read()
print (s)
f.close()
```

El archivo .txt esta guardados en el mismo directorio que el ejecutable .py o puede estar almacenado en cualquier otra ubicación. Solo es necesario definir la ruta en donde se encuentra guardado.

Python – Listas

Entre las *secuencias*, el más versátil, es la *lista*, para definir una, se debe escribir entre corchetes, separando sus elementos con comas cada uno.

La lista en Python son variables que almacenan arrays, internamente cada posición puede ser un tipo de datos distinto.

```
factura = ['pan', 'huevos', 100, 1234]  
print(factura)
```

- Heterogéneas: pueden estar conformadas por elementos de distintos tipo, incluidos otras listas.
- Mutables: sus elementos pueden modificarse.

Una lista en Python es una estructura de datos formada por una secuencia ordenada de objetos.

Los elementos de una lista pueden accederse mediante su índice, siendo 0 el índice del primer elemento

Python – Listas

Ejemplo:

```
>>> factura[0]  
'pan'  
>>> factura[3]  
1234
```

La función ***len()***devuelve la longitud de la lista (su cantidad de elementos).

```
>>> len(factura)  
4
```

Se puede obtener el elemento de una lista usando los mismo métodos visto con los String

```
>>> factura[-1]  
1234
```

Python – Listas: Métodos

El objeto de tipo *lista* integra una serie de métodos que se describen a continuación:

- **append()**: Este método agrega un elemento al final de una lista.
- **count()**: Este método recibe un elemento como argumento, y cuenta la cantidad de veces que aparece en la lista.
- **extend()**: Este método extiende una lista agregando un iterable al final.
- **index()**: Este método recibe un elemento como argumento, y devuelve el índice de su primera aparición en la lista. El método devuelve un excepción *ValueError* si el elemento no se encuentra en la lista, o en el entorno definido.
- **insert()**: Este método inserta el elemento x en la lista, en el índice i.
- **pop()**: Este método devuelve el último elemento de la lista, y lo borra de la misma.
- **remove()**: Este método recibe como argumento un elemento, y borra su primera aparición en la lista.
- **reverse()**: Este método invierte el orden de los elementos de una lista.

Python – Diccionarios

Diccionarios de Python son una lista de consulta de términos de los cuales se proporcionan valores asociados.

En Python, un diccionario es una colección no-ordenada de valores que son accedidos a través de una clave. Es decir, en lugar de acceder a la información mediante el índice numérico, como es el caso de las listas y tuplas, es posible acceder a los valores a través de sus claves, que pueden ser de diversos tipos.

Las claves son únicas dentro de un diccionario, es decir que no puede haber un diccionario que tenga dos veces la misma clave, si se asigna un valor a una clave ya existente, se reemplaza el valor anterior.

No hay una forma directa de acceder a una clave a través de su valor, y nada impide que un mismo valor se encuentre asignado a distintas claves.

Cualquier variable de tipo inmutable, puede ser clave de un diccionario: cadenas, enteros, tuplas (con valores inmutables en sus miembros), etc. No hay restricciones para los valores que el diccionario puede contener, cualquier tipo puede ser el valor: listas, cadenas, tuplas, otros diccionarios, objetos, etc.

Python – Diccionarios

De la misma forma que con listas, es posible definir un diccionario directamente con los miembros que va a contener, o bien inicializar el diccionario vacío y luego agregar los valores de a uno o de a muchos.

```
punto = {'x': 2, 'y': 1, 'z': 4}
```

Para declararlo vacío y luego ingresar los valores, se lo declara como un par de llaves sin nada en medio, y luego se asignan valores directamente a los índices.

```
facturacion = {}
facturacion["lunes"] = [6103, 7540]
facturacion["martes"] = [6201]
facturacion["miércoles"] = [6103, 7540]
facturacion["jueves"] = []
facturacion["viernes"] = [6201]

print(facturacion)
print(facturacion["lunes"])
```

Python – Diccionarios

Ejemplo:

```
facturacion = {}

facturacion["lunes"] = [6103, 7540]
facturacion["martes"] = [6201]
facturacion["miércoles"] = [6103, 7540]
facturacion["jueves"] = []
facturacion["viernes"] = [6201]

print(facturacion)
print(facturacion["lunes"])

lista=facturacion["lunes"]

for i in range(0,len((lista))):
    valor=lista[i]
    print(valor)
```

Python – Diccionarios

Ejemplo con funciones:

```
datos={}
datos["NOMBRE"]=""
datos["APELLIDO"]=""
datos["EDAD"]=""  
  
base_datos=[]  
  
def datos_personales():
    for i in range(0,2):
        print("INGRESE EL NOMBRE")
        nombre=input()
        print("INGRESE EL APELLIDO")
        Apellido=input()
        print("INGRESE LA EDAD")
        Edad=input()  
  
        datos["NOMBRE"]=nombre
        datos["APELLIDO"]=Apellido
        datos["EDAD"]=Edad  
  
    base_datos.append(datos)  
  
return(base_datos)
```

```
base=datos_personales()
print(base)
```