Python

A lo largo de esta aventura de aprendizaje te mostramos cómo emplear Python para el desarrollo de programas. Iniciamos con una **introducción al lenguaje de programación** que incluye sus características principales. Para luego incursionar en la estructura de este lenguaje, iniciando por los **tipos de datos**.

Luego pasamos a explorar las variables para el almacenamiento de datos de diferentes tipos. Seguimos con las instrucciones en Python para la **captura de datos (input())** y el **despliegue de los mismos vía la consola (print())**.

A continuación te proporcionamos una serie de **módulos predefinidos** y **funciones predefinidas (e.g. open())**.

Nuestra siguiente parada fue por la estructura de control para el soporte de **bifurcaciones (if, else)** y para la ejecución de **ciclos (for y while)**. Y luego exploramos las **estructuras de datos de listas, diccionarios, tuples y ranges**.

El siguiente turno fue para las **funciones** que proveen soporte a un código más estructurado basado en el reuso. Python proporciona alternativas para la **programación orientada a objetos** que también aplicamos en nuestro proceso de aprendizaje.

Hoy en día nuestros programas pueden acceder información desde varios sitios en Internet. Es por ello que aplicamos los módulos de **Python para el soporte de Servicios Web** para la obtención de data de diferentes sitios Web.,

El uso de base de datos es necesario para el almacén de datos de forma estructurada. Examinamos la forma de conectarnos, acceder y manejar una base de datos **MySQL y Postgres** desde un código Python.

Nuestro trayecto termina con las **expresiones regulares** para validar ciertos patrones y la manipulación de errores y fallas en nuestro programa, a través del **manejo de excepciones**.

Bajo el enfoque teórico-práctico, has ido obteniendo las habilidades necesarias para construir tus programas Python.

Luego de haber culminado exitosamente el curso de Python, has desarrollado las siguientes capacidades:

- Resumir la sintaxis y semántica de Python para el desarrollo de programas.
- Crear un programa en Python usando los diversos componentes del lenguaje de programación: tipos de datos, estructuras de datos, funciones, expresiones regulares y estructuras de control.

- Desarrollar un programa en Python usando los componentes para la programación orientada a objetos que proporciona el lenguaje.
- Crear un programa en Python que interactúe con una base de datos MySQL o Postgres.
- Desarrollar un programa en Python que capture datos usando un Servicio Web.
- Crear un programa en Python que maneje errores o fallos usando excepciones.
- Aplicar multiprocesamiento en el desarrollo de códigos en Python.

Introducción a Python

Python es un lenguaje de programación de propósito general. Creado por Guido van Rossum, a finales de la década de los 80 y cuya primera versión fue publicada en los 90. Actualmente está **disponible la versión 3** de este lenguaje de programación.

Entre las **características más importantes de Python** se tienen las siguientes:

- Es de propósito general
- Es un lenguaje interpretado
- Posee una sintaxis clara y de alto nivel
- Está disponible en múltiples sistemas operativos y plataformas
- Dispone gran cantidad de código reutilizable

Tipos de Datos

Entre los **tipos de datos básicos** que se manejan en Python se tienen:

- Los números enteros
- Los números flotantes o números reales
- Las cadenas de caracteres o strings
- Lógicos o booleanos

Dependiendo del tipo de dato se tienen **diversos operadores**, los cuales se pueden aplicar sobre estos tipos. Por ejemplo, los tipos de datos numéricos, como son los enteros y los reales, tienen operadores de suma, resta, multiplicación y división.

Variables y Uso de Entrada y Salida de Datos

Python es un **lenguaje de tipificado** dinámico, que asigna dinámicamente el tipo de datos a una variable o constante, dependiendo del valor que se le asocie (ver Figura 1).

Cuando definimos variables, se recomienda utilizar siempre el nombre en minúsculas y si va a estar conformada por varias palabras, se sugiere usar como separador de palabras, el símbolo de guión bajo.

```
>>> x = 1
>>> type (x)
<class 'int'>
>>> x = 1.2
>>> type (x)
<class 'float'>
>>> x = "Hola"
>>> type (x)
<class 'str'>
```

Figura 1: Ejemplo del uso de variables en Python.

Operaciones de Lectura y Escritura

La **instrucción print** se emplea para desplegar un mensaje específico en la pantalla u otro dispositivo de salida.

La **instrucción input** se emplea para capturar datos de entrada.

```
>>> letra = "A"
>>> letra = input ("Por favor ingresa una letra: ")
Por favor ingresa una letra: C
>>> print ("El usuario ingresó esta letra:" + letra)
El usuario ingresó esta letra:C
```

Figura 2: Ejemplo del uso de las instrucciones de lectura y escritura en Python.

Módulos y Funciones

Python ofrece una serie de módulos predefinidos que soportan características de gran utilidad al momento de resolver problemas. Un módulo contiene definiciones de funciones y opcionalmente puede contener instrucciones para inicializar el módulo. Un módulo es guardado como un archivo **nombre_modulo.py**. A continuación algunos módulos predefinidos en Python:

- **math:** Funciones matemáticas. Es importante destacar que este módulo siempre está disponible, por lo cual para usarlo, sólo hay que colocar import math.
- **urllib, smtplib, poplib, email:** Módulos para funcionalidades de diversos protocolos de internet.
- **datetime, timeit:** Manejo de fecha y hora.

Por ejemplo, una vez que importamos el **módulo datetime** podemos emplear las **clases date y datetime**. La **clase date** permite obtener la fecha del sistema, mientras que la **clase datetime** permite generar la fecha y hora del sistema. Por ejemplo el siguiente código

```
from datetime import date, datetime
from dateutil.parser import parse
from dateutil.relativedelta import relativedelta
fecha = date.today()
fecha_hora = datetime.now()
print("Formato de Fechas")
print("Fecha:",fecha)
```

Genera el siguiente resultado:

```
Formato de Fechas
Fecha: 2021-11-13
```

Funciones Predefinidas Archivos

Las siguientes **funciones predefinidas** permiten manejar archivos en Python:

- **open():** abre un archivo para lectura (defecto) o escritura. Puede crear un archivo, si se especifica.
- **read():** permite leer el contenido completo de un archivo.
- **readline():** permite leer una línea de un archivo.

- **seek():** nos permite ubicarnos en una posición específica en un archivo.
- **readlines():** retorna una lista conteniendo cada línea del archivo.
- **write():** permite escribir en un archivo
- **writable():** retorna verdadero, si el archivo se puede escribir.

El siguiente código genera un archivo llamado **ejemplo.txt**, escribe una línea de texto en dicho archivo, lee la línea del archivo e imprime lo que leyó.

```
with open("ejemplo.txt",'w') as archivo1:
  archivo1.write ("Una línea")
with open("ejemplo.txt",'r') as archivo1:
    linea = archivo1.readline()
print (linea)
```

El código anterior genera el siguiente resultado:

```
Una línea
```

Estructuras de Control

Para realizar acciones condicionadas en Python, hacemos uso de la **instrucción if**. Las instrucciones que están dentro un **if** deben estar indentadas. Si este condicional no es cierto, entonces podemos utilizar la **instrucción else**, para indicar qué otras acciones queremos ejecutar. También podemos **emplear if** y **elif** para verificar que se cumpla ciertas condiciones en cada una de las instrucciones.

El siguiente código:

```
num = input ("Por favor, ingrese un número: ")
num1 = int (num)
if num1 > 200:
    print ("El número es mayor que doscientos")
else:
     print ("El número es menor o igual que doscientos")
```

Genera el siguiente resultado si el usuario ingresa el número 200

```
Por favor, ingrese un número: 200
El número es menor o igual que doscientos
```

El siguiente código:

```
num = input ("Por favor, ingrese un número: ")
num1 = int (num)
if num1 > 200:
    print ("El número es mayor que doscientos")
elif num1 > 100:
      print ("El número es menor que doscientos pero mayor que
cien")
else:
     print ("El número es menor o igual que cien")
```

Genera el siguiente resultado si el usuario ingresa el número 150.

```
Por favor, ingrese un número: 150
El número es menor que doscientos pero mayor que cien
```

La **instrucción for** se usa para iterar sobre una secuencia de instrucciones. Por ejemplo, en el siguiente código se emplea la **instrucción for** para imprimir las ciudades en una lista de ciudades de Aotearoa, Nueva Zelanda.

```
cities = ["Auckland","New Plymouth","Masterton"]
for city in cities:
  print (city)
```

Genera el siguiente resultado por pantalla:

```
Auckland
New Plymouth
Masterton
```

En los **ciclos for y while**, así como en los condicionales siempre se puede detener la ejecución de su propio bloque de instrucciones con **break**.

Estructuras de Datos

La **lista** es una estructura de datos que contiene elementos de cualquier tipo encerrados entre corchetes y separados por comas. En una lista se puede tener elementos de tipo string, de tipo numérico, o cualquier otro tipo de dato, incluyendo el tipo lista. A continuación algunos ejemplos de listas:

```
lista_de_enteros = [1,2,3,4]
saludos_maori = ["Kia ora","Morena","Kia ora koutou"]
lista_de_listas = [[23,25,26],[1.2,1.3,1.4,1.5]]
```

Los **diccionarios** son usados para almacenar datos en la forma de **clave:valor**. Los elementos de un diccionario se colocan entre llaves y están ordenados y no se aceptan duplicados.

El siguiente código imprime el contenido del **diccionario ciudad**:

```
ciudad =   {
  "nombre": "Auckland",
  "region": "Auckland"

}
print (ciudad)
```

Resultado:

```
{'nombre': 'Auckland', 'region': 'Auckland'}
```

Las **tuples** son secuencias inmutables normalmente utilizadas para almacenar información heterogénea. El hecho de que sea inmutable hace que sus valores no puedan ser modificados, pero si almacenan tipos de datos como listas, estas pueden ser modificadas.

El siguiente código imprime el contenido la **tuple a**:

```
a="Pera","Rojo",3
print (a)
```

Resultado

```
('Pera', 'Rojo', 3)
```

Los **ranges** son secuencias inmutables de números, tal como se muestra a continuación:

```
a=range(10)
print (list(a))
```

Resultado

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Funciones y Orientado a Objeto

Una función es un bloque de código que puede pasar cero o más parámetros y que sólo se ejecuta cuando se llama. Una función puede o no retornar un resultado.

```
def una_funcion():
  print("Soy una función")
una_funcion()
```

El resultado de ejecutar el código anterior es:

```
Soy una función
```

En Python podemos representar casi todo como un **objeto con sus propiedades y métodos**. Usamos el **constructor de objetos class** para crear objetos. A continuación un ejemplo donde definimos la **clase Perro**, que tiene dos **métodos Ladra y Busca**. Definimos un objeto de la **clase Perro que es perr1**.

```
class Perro:
    def __init__(self, color, raza, nombre):
        self.color = color
        self.raza = raza
        self.nombre = nombre

    def Busca (self,item):
        print(self.nombre + " busca " + item)

    def Ladra (self):
        print (self.nombre, " ladra ")

perr1= Perro ("blanco","Bulldog","Perrito")
perr1.Ladra()
perr1.Busca("pelota")
```

El código anterior genera lo siguiente:

```
Perrito  ladra
Perrito busca pelota
```

En Python existe la **herencia de clases** que permite derivar clases desde una

clase base. La ejecución de una definición de clase derivada procede de la misma manera que para una clase base. Cuando se construye el objeto de clase, también se hace uso de la clase base. Esta relación con la clase base, se utiliza para resolver referencias de atributos: si un atributo solicitado no se encuentra en la clase derivada, se procede a buscar en la clase base. Esta regla se aplica de forma recursiva, si la clase base en sí, se deriva de alguna otra clase.

Conexión con Internet

El **uso de Servicios Web** bajo desarrollos en Python es muy sencillo, sólo se tiene que emplear la librería requests y seleccionar cuál es el método de conexión http a utilizar, pasando los parámetros correspondientes: URL , headers y parameters. El resultado obtenido se puede tratar como un JSON. Los elementos del documento JSON se tratan como un diccionario donde la clave es el nombre del atributo y el valor es el valor correspondiente a ese atributo.
Seguidamente encontrarás un ejemplo:

```
import requests

url =
'https://pro-api.coinmarketcap.com/v1/cryptocurrency/listings/lat
est'

headers = {
  'Accepts': 'application/json',
  'X-CMC_PRO_API_KEY': '7a138f90-8d38-49cb-84a4-01a98c978996',
}

parameters = {
  'limit':'10',
  'convert':'USD',
  'sort':'volume_24h'
}

result=requests.get(url,headers=headers, params=parameters)
cripto_lastest=result.json()['data']
```

Módulos de Conexión a Internet - Protocolos

Con la cantidad de información y datos que existe en Internet siempre será necesario y de gran utilidad, acceder o colocar información a través de Internet, por lo cual es importante conocer algunos de los módulos que permiten a Python usar o colocar dicha información. Algunos módulos de conexión a Internet se listan a continuación:

- **módulo smtplib:** que permite manejar conexión del protocolo SMTP, permitiendo, entre otros, enviar correos electrónicos.

- **módulo ftplib:** permite manejar el cliente del protocolo FTP, tal como establecer una conexión vía FTP hacia un servidor FTP para obtener la información requerida.

Python y Bases de Datos

Python ofrece unas librerías para manipular la base de datos **MySQL y PostgreSQL: MySQL y psycopg2**, respectivamente.

En general, para trabajar con una base de datos MySQL en Python, se debe realizar lo siguiente:

- Establecer la conexión con una base de datos empleando mysql.connector.connect.
- Instanciar un cursor para poder recorrer los resultados de sentencia SQL realizada a una base de datos, usando db.cursor() (asumiendo que la base de datos es **db**)
- Ejecutar la instrucción SQL empleando cur.execute (asumiendo que el cursor está en **cur**)
- Asegurar que los cambios en la base de datos son consistentes empleando db.commit (asumiendo que la base de datos es db).

A continuación un ejemplo:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="nombreusuario",
  password="contrasena",
  database="basededatos"
)

mycursor = mydb.cursor()

sql = "INSERT INTO clientes (nombre, apellido) VALUES (%s, %s)"
val = ("Jose", "Perez")

mycursor.execute(sql, val)

mydb.commit()
```

Para usar una **base de datos PostgreSQL**, importamos el **módulo psycopg2** que tiene las librerías necesarias para utilizar los ODBC de conexión con la base de datos PostgreSQL.

```
import psycopg2
```

Empleamos el **método connect** para conectar con la base de datos. En el siguiente ejemplo, **conf** contiene la configuración de la base de datos:

```
mydb = psycopg2.connect(
  host=conf.database_conn['host'],
  user=conf.database_conn['user'],
  password=conf.database_conn['password'],
  database=conf.database_conn['database'],
  port='5433'
```

Entonces asociamos un cursor que nos permita hacer el recorrido por la base de datos usando **cursor**:

```
mycursor = mydb.cursor()
```

Ejecutamos la instrucción SQL de acuerdo a la operación que deseamos aplicar a la base de datos usando **execute**:

```
sql="INSERT INTO coin_price (timestamp, c1oin_id, price) VALUES
(%s,%s,%s)"
val=(datetime.now(),x[0],coin['precio'])
 mycursor.execute(sql, val)
```

Aseguramos que los cambios en la base de datos son consistentes empleando **commit**.

```
mydb.commit()
```

Expresiones Regulares y Manejo de Excepciones

Una **expresión regular** está formada por una secuencia de caracteres que forman un patrón para validar otra cadena de caracteres. La siguiente expresión regular chequea que una contraseña cumple con la siguiente política. La política contempla que la contraseña debe tener al menos una letra en minúscula, una letra en mayúscula, un dígito numérico y caracteres especiales.

```
password_pattern="(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&/_=;
:<>|#,\*\'\"\-\+\(\)\{\}\.\+\[\]])[A-Za-z\d@$!%*?&/_=;:<>|#,\*\'\
"\-\+\(\)\{\}\.\+\[\]]{8,}$"
```

Manejo de Excepciones

Un código en Python puede generar errores en el tiempo de ejecución. Estos errores o excepciones se pueden capturar para tratarlos apropiadamente, logrando así que estos fallos no afecten directamente la ejecución del programa.
Las instrucciones **try ... except** se pueden emplear para capturar y procesar las excepciones o fallos. A continuación un ejemplo de la sintaxis.

```
try:
    cantidad=int(input('Indique la cantidad de Passwords a generar: '))
  except ValueError:
       print('El valor ingresado es inválido. Debe ingresar un número
entero')
```

Multiprocesamiento

Para manejar múltiples procesos ejecutándose concurrentemente en un programa empleamos el **módulo multiprocessing**, que forma parte de las **librerías standards** de Python y a su vez, importamos la **librería pool**. Con esta librería, se logra crear un **pool de procesos** que pueden ser usados en nuestros códigos.

```
from multiprocessing import Pool

myPool = Pool(2)
```

La **función apply_async** nos permite hacer la creación o invocación de un nuevo proceso dentro del pool (ver código a continuación). El **argumento callback** de esta función señala, una vez que termine la ejecución de la función en el argumento de **apply_async** (e.g., get_data), qué función de nuestra aplicación principal se va a ejecutar (e.g., fin_data).

```
result=myPool.apply_async(get_data,args=(time,),callback=fin_data
)
```

Módulo curses

El **módulo curses** nos permite manipular la interfaz gráfica para poder desplegar la información en dicha interfaz de una forma determinada, superando así las restricciones que tenemos cuando empleamos las **instrucciones input y print**.

A continuación se describen algunos métodos de este módulo:

- **curses.noecho():** se utiliza para evitar que se muestre en la pantalla las teclas que se pulsen.
- **curses.cbreak():** permite que el programa reaccione ante el evento de pulsar una tecla, lo que hace que cualquier tecla se comporte como la tecla Enter o Intro.
- **stdscr.keypad(True):** se utiliza para que **curses** genere los valores de retorno de las teclas especiales, tales como: las fechas, paginación, entre otras.

- **curses.newwin():** permite crear una nueva ventana, pudiendo colocarla en una sesión de la ventana principal creada con la función initscr().

En **curses**, hay que tener muy en cuenta que el sistema de coordenadas es diferente al tradicional. Las coordenadas siempre se pasan en el **orden (y,x)**, y la esquina superior izquierda de una ventana es la **coordenada (0,0)**.