Como abrir extensions: (CTRL+MAYUSC+X)

Como abrir explorer: (CTRL+MAYUSC+E)

Folder-Workspace-File

Intellisense (Ayuda a terminar o dictar la función que quieres utilizar) (CTRL+BARRA ESPACIADORA)

Code Snippets: Facilita para repetir patrones de codigos como loops o estados de condición.

Mandar a correr un código: (CTRL+ALT+N)

Como guardar un archivo: (CTRL+S)

Guardar como: (CTRL+MAYUSC+S)

PYTHON GENERALIDADES:

- **Propósito Generales:** Permite desarrollar aplicaciones de cualquier tipo de rama del conocimiento y cualquier tipo de aplicación.
- **Tercera Generación:** Es llamado así, ya que esta basado en un conjunto de instrucciones que permiten detallar de forma procedural y secuencial la forma en como han de realizarse las tareas.
- **Orientado a objetos:** Permite la creación e instanciación de clases, permite implementar las tres características POO:
    1. Encapsulamiento: Significa reunir todos los elementos que pueden considerarse pertenecientes a una misma entidad. Al mismo nivel de abstracción.
    2. Herencia: Las clases no están aisladas, sino que se relacionan entre si, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. Organiza y facilita el polimorfismo y encapsulamiento.
    3. Polimorfismo: Comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre, pero al llamarlos se utilizará el comportamiento correspondiente al objeto que se esté usando.
- **Interpretado:** Cada vez que se manda a ejecutar un programa, se realiza la verificación sintáctica (parsing) y la generación de código binario ejecutable, para la plataforma en que es ejecutado (compilación).
- **Sensible a mayúsculas y minúsculas (case sensitive):** Si importa el uso de mayúsculas y minúsculas, para palabras reservadas, como para el nombre de elementos creados por el desarrollador.
- **Posicional (Indented):** La posición en la que inicia una línea de código influye en la manera en la que el interprete de Python, produce resultados.
- **No requiere la especificación detallada de tipos (Non Strong Type):** Las variables no requieren ser declaradas antes de utilizarse, ni deben utilizarse si son declaradas.
- **Sin terminador de línea:** No requiere de “FinMientras” etc.

PYTHON TUTORIAL

- Cuál es el terminador de línea en Python (**Enter**).
- Cómo defino un bloque de código en Python (*Identacion*).
  ```
  If 5>2:
    print (“El 5 es mayor a 2”)
  ```
- Cómo se colocan comentarios en Python (**#**).
  ```
  #Este programa sirve de tal y tal forma.
  ```
- Cómo genero una salida de consola en Python (**print**).
  ```
  print (“Hello World”)
  ```
- Cómo se definen las variables en Python.
  X = “Hola” o Y = ‘hola’
- Funciones para el establecimiento explícito de *data type*.

Example	Data Type
`x = str("Hello World")`	str
`x = int(20)`	int
`x = float(20.5)`	float
`x = complex(1j)`	complex
`x = list(("apple", "banana", "cherry"))`	list
`x = tuple(("apple", "banana", "cherry"))`	tuple
`x = range(6)`	range
`x = dict(name="John", age=36)`	dict
`x = set(("apple", "banana", "cherry"))`	set
`x = frozenset(("apple", "banana", "cherry"))`	frozenset
`x = bool(5)`	bool
`x = bytes(5)`	bytes
`x = bytearray(5)`	bytearray

```
x = memoryview(bytes(5))
```
memoryview

- Cuáles son las reglas de nombrado de variable en Python.
    1. El nombre de una variable debe iniciar con una letra o guion bajo.
    2. No puede iniciar con un número.
    3. Solo puede tener caracteres alfanuméricos y guion bajos.
    4. Las variables son sensibles, “edad”, “Edad” y “EDAD” son diferentes.

- Qué tipos de notación son los más utilizados en la actualidad para el nombrado de elementos en un programa:

  NombreUsuario - Pascal Casing (Todos los inicios de palabra en mayúsculas)
  nombreUsuario - Camel Casing (Primera minúscula, y luego mayúsculas los inicios de palabra.
  objLista – Hungarian notation (Incluye referencia al tipo, como prefijo o como sufijo)

- Qué notación se sugiere para el nombrado de variables en Python (*camelCase*).
- Qué notación se sugiere para el nombrado de clases, métodos y funciones (*PascalCase*).
- Cómo se declaran funciones simples, que no reciben parámetros, y no retornan valores (**def**).
  ```
  def my_function():
    print("Hello from a function")
  ```
- Qué son las variables globales.
  Variables that are created outside of a function (as in all of the examples above) are known as global variables
- Que son las variables locales.
  If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function.
- Para que sirve **global**
  If you use the global keyword, the variable belongs to the global scope
- Cuáles son los *data types* admitidos por Python
  - Text Type: `str`
  - Numeric Types: `int`, `float`, `complex`
  - Sequence Types: `list`, `tuple`, `range`
  - Mapping Type: `dict`
  - Set Types: `set`, `frozenset`
  - Boolean Type: `bool`
  - Binary Types: `bytes`, `bytearray`, `memoryview`

- Función para determinar el tipo de dato de una variable (**type()**)
  Print(Type(x))
- Funciones de conversión de tipos de datos.
  ```
  a = float(x)

  #convert from float to int:
  b = int(y)

  #convert from int to complex:
  ```

```
c = complex(x)
```

- Uso de **str.isdigit()**
  Returns True if all characters in the string are digits
- Cómo se importan librerías a un programa Python (**import**)

- Función de generación de números aleatorios (**random.randrange()**). Requiere importar la librería (**random**).
- Función para preguntar información desde la consola (**input()**)
- Trabajo con strings.
  - Uso de literal.
    1. are surrounded by either single quotation marks, or double quotation marks.
  - Uso de multilínea.
    2. You can assign a multiline string to a variable by using three quotes:
       ```
       a = """Lorem ipsum dolor sit amet,
       consectetur adipiscing elit,
       sed do eiusmod tempor incididunt
       ut labore et dolore magna aliqua."""
       print(a)
       ```
  - Uso de **str.len**.
    The `len()` function returns the length of a string:
    ```
    a = "Hello, World!"
    print(len(a))
    ```
  - Métodos:
    - **str.strip**

      The `strip()` method removes any whitespace from the beginning or the end:
      ```
      a = " Hello, World! "
      print(a.strip()) # returns "Hello, World!"
      ```
    - **str.lower:**
      The `lower()` method returns the string in lower case:
      ```
      a = "Hello, World!"
      print(a.lower())
      ```

- **str.upper**
  The `upper()` method returns the string in upper case:
  ```
  a = "Hello, World!"
  print(a.upper())
  ```

- **str.replace**

  The `replace()` method replaces a string with another string:

  ```
  a = "Hello, World!"
  print(a.replace("H", "J"))
  ```

- **str.split**
  The `split()` method splits the string into substrings if it finds instances of the separator:
  ```
  a = "Hello, World!"
  print(a.split(",")) # returns ['Hello', ' World!']
  ```

- **str.partition**

[partition()]	Returns a tuple where the string is parted into three parts

    - **not in**
      ```
      txt = "The rain in Spain stays mainly in the plain"
      x = "ain" not in txt
      print(x)
      ```
    - Concatenación
      ```
      a = "Hello"
      b = "World"
      c = a + b
      print(c)
      ```
- Dando formato a strings usando placeholders (**str.format**)
  The `format()` method formats the specified value(s) and insert them inside the string's placeholder.
  string.format(*value1, value2...*)
- Manejo de datos booleanos.
  You can evaluate any expression in Python, and get one of two answers, `True` or `False`.
  ```
  print(10 > 9)
  print(10 == 9)
  print(10 < 9)
  ```

- Manejo de operadores (aritméticos, asignación, lógicos, comparación)
  - Sólo los más usuales.
  - Especial atención en el uso de **`is`** e **`is not`**, con **`type()`**.
- Manejo de **`if:`**
- Manejo de **`if`** / **`else:`**
- Manejo de **`while:`**
- Manejo de **`while:`** / **`else:`**
- Uso de **`break`** (para ciclos infinitos)

  With the `break` statement we can stop the loop even if the while condition is true:

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

- Manejo de **`for`**

  A `for` loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string)

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

- Uso de **`range()`** con **`for.`**

  The `range()` function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

```
for x in range(6):
  print(x)
for x in range(2, 30, 3):
  print(x)
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```