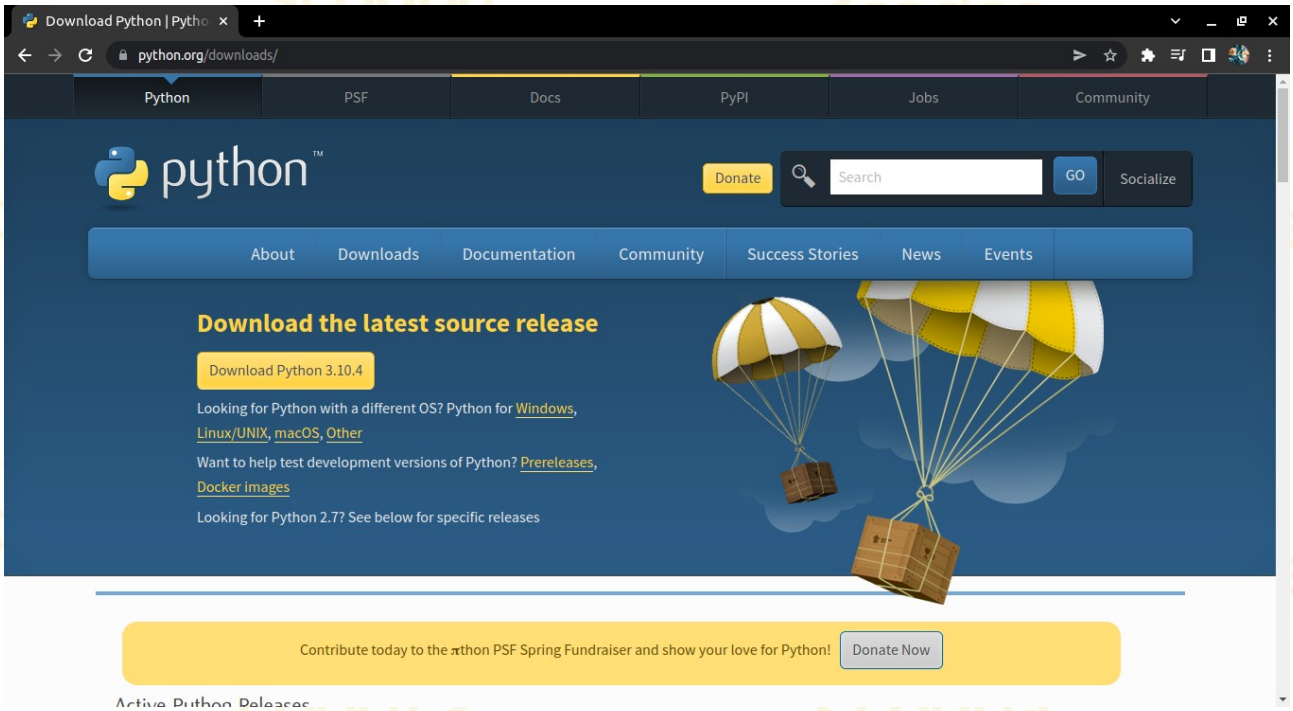
UNIDAD I : Primeros pasos en Python

Instalar Python en Windows y MacOS

En ambos sistemas operativos la instalación es muy similar.



1. Ir a la página oficial de python https://www.python.org/downloads/ y descargar la última versión dándole click al botón amarillo.
2. Una vez descargado el archivo, ejecutarlo (con permisos de administrador)
3. Seleccionar Install launcher for all users y Add Python to PATH (importante!)
4. Iniciar la instalación
5. Una vez terminado el proceso, cerrar el mensaje de notificación.

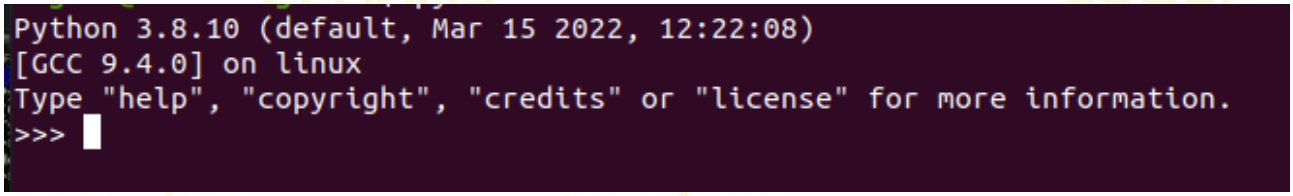
Probando la instalación (Windows)

1. En el botón de inicio, Ejecutar, escribir cmd. Se abrirá la línea de comandos en una ventana negra.
   Allí tipear
   ```
   python --version
   ```
   Si todo está correcto, aparecerá la versión de python que hemos instalado.
2. Si escribimos
   ```
   python
   ```
   se nos abrirá la consola/intérprete de python, que podremos distinguir por el prompt `'>>>'`.
   Allí podemos tipear
   ```
   print('Hola mundo!')
   ```
   y habremos escrito nuestra primera línea en python.

El intérprete de Python

El intérprete funciona de manera similar al shell de Unix: cuando se le llama con una entrada estándar conectada a un terminal, lee y ejecuta comandos de manera interactiva; cuando se le llama con un argumento de nombre de archivo o con un archivo como entrada estándar, lee y ejecuta un script desde ese archivo.

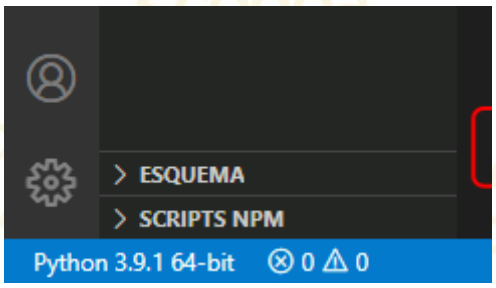
Podemos salir del intérprete, tipeando `quit()`



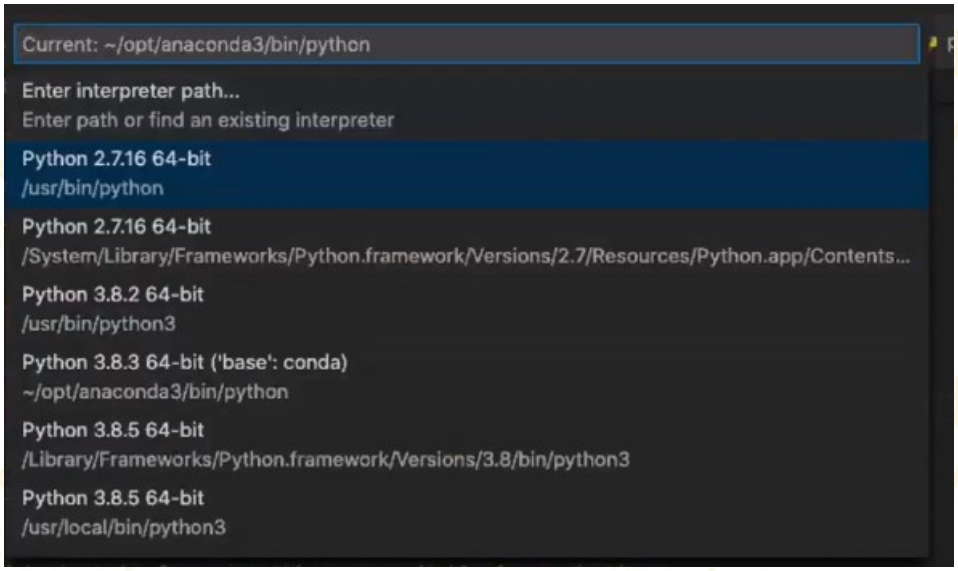
Instalación en Visual Studio Code

En la pestaña de extensiones, buscar Python. Insalamos la extensión de Microsoft.

Si tenemos varias versiones del intérprete Python, podemos seleccionar cuál queremos usar, pulsando en la barra azul, abajo a la izquierda.



En la parte superior se nos va a desplgar un menú en el que podremos elegir la versión que queremos utilizar.



PIP

Es el instalador de paquetes de Python. Es una utilidad que se ejecuta desde la línea de comandos, que permite instalar, reinstalar o desinstalar paquetes de python.

En versiones superiores s 2.79. o 3.4, PIP ya viene instalado con Python por defecto.

Enlaces externos

Página oficial de descargas Python: https://www.python.org/downloads/

Documentación oficial python en español: https://docs.python.org/es/3/

Ambientes virtuales: Virtualenv

Los ambientes virtuales nos permiten encapsular un proyecto para poder instalar versiones de los paquetes que se requieran sin tener que instalarlos en todo el sistema operativo. Esto evita que se produzcan conflictos de paquetes en el intérprete principal.

Para trabajar en entornos virtuales:

1. Instalar el paquete de virtualenv de forma global, ejecutando
   `pip install virtualenv`
   Para verificar que se instaló correctamente podemos ejecutar `which virtualenv`
2. Crear o seleccionar la carpeta en donde tendremos nuestro entorno virtual, estando ahí ejecutamos lo siguiente para crear el entorno virtual:
   `virtualenv nombre_entorno`
   por convención es recomendable llamarlo `venv`
3. Después de crear el entorno virtual debemos activarlo, para ello se ejecuta el siguiente comando:
   `source /venv/Scripts/activate` (para windows)
   `source /venv/bin/activate` (en linux)
   con esto quedará activado y nos aparecerá el nombre del entorno virtual al inicio de la línea en la terminal de comandos, (venv en este caso).
   Para desactivarlo sería lo mismo pero al final se coloca
   `deactivate`
4. Para ver los paquetes que tenemos instalados en nuestro entorno virtual ejecutamos el siguiente comando:
   `pip freeze`
   Esto nos mostrará el listado de los paquetes, si no aparece nada es porque no se ha instalado ningún paquete aún.
5. Si queremos tener todos los paquetes agrupados y con su versión correspondiente, podemos hacer uso del archivo requirements.txt en donde colocaremos cada uno de los paquetes
   `pip freeze > requirements.txt`
   Luego podremos distribuir nuestro trabajo y quienes quieran trabajar sobre él, podrán instalar los paquetes requeridos usando el siguiente comando:
   `pip install -r requirements.txt`

```
regina@UbuntuRegina:~$ virtualenv venv
created virtual environment CPython3.8.10.final.0-64 in 4026ms
  creator CPython3Posix(dest=/home/regina/venv, clear=False, no_vcs_ignore=False
, global=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle
, via=copy, app_data_dir=/home/regina/.local/share/virtualenv)
    added seed packages: pip==21.3.1, setuptools==59.6.0, wheel==0.37.0
  activators BashActivator,CShellActivator,FishActivator,NushellActivator,PowerS
hellActivator,PythonActivator
regina@UbuntuRegina:~$ source venv/bin/activate
(venv) regina@UbuntuRegina:~$ deactivate
regina@UbuntuRegina:~$
```

Tipos de datos en Python

Los tipos de datos se pueden clasificar en:

Mutable: su valor puede cambiarse en tiempo de ejecución.

Inmutable: su valor no puede cambiarse en tiempo de ejecución.

Enteros (int)

Representan todos los números enteros (positivos, negativos y 0), sin parte decimal.

Reales (float)

Representa los números reales, tienen una parte entera y otra decimal.

Complejos (complex)

Representan números complejos, con una parte real y otra imaginaria. Son tipos de datos inmutables.

Booleanos

Pueden tener únicamente dos tipos de valores: "True" o "False". Estos sirven para agregar atributos a la variable si es verdadero o falso.

Cadena (str)

Permite almacenar cadenas de caracteres. Es de tipo inmutable.

Lista (list)

Variable que almacenan arrays, internamente cada posición puede ser un tipo de datos distinto. Se escribe entre corchetes, y sus elementos se separan con comas.
Ejemplo: `ingredientes = ['huevos', 'harina', 'leche', 'azúcar']`

Tupla

Las tuplas son objetos de tipo secuencia, específicamente es un tipo de dato lista inmutable. Esta no puede modificarse de ningún modo después de su creación.
Ejemplo: `mi_tupla = ('Python', True, 'ananá', 34)`

Diccionario (dict)

Pueden ser creados colocando una lista separada por coma de pares "key:value" entre {}. Único tipo de mapeo estándar actual.
Ejemplo: `mi_dict = {'Ana': 27, 'Juan': 51}`

Conjuntos

Un conjunto, es una colección no ordenada y sin elementos repetidos. Los usos básicos de éstos incluyen verificación de pertenencia y eliminación de entradas duplicadas.

set: Mutable, sin orden, no contiene duplicados.
Ejemplo: `my_set = set([3.6, 'Entre Ríos', False])`

frozenset: Inmutable, sin orden, no contiene duplicados.
Ejemplo: `my_fs = frozenset([3.6, 'Entre Ríos', False])`

UNIDAD II: Operadores en Python

De asignación

Se utilizan para asignar valores a una variable.

b = 5	la variable b vale 5
b += 5	equivale a b = b + 5
b -= 5	equivale a b = b - 5
b *= 5	equivale a b = b * 5
b /= 5	equivale a b = b / 5
b %= 5	equivale a b = b % 5
b **= 5	equivale a b = b ** 5
b //= 5	equivale a b = b // 5

Matemáticos

Son todos aquellos que nos permiten a hacer cualquier operación básica.

+	suma
-	resta
*	multiplicación
/	división
%	módulo
**	potencia
//	división entera

Relacionales

Se utilizan para comparar y establecer la relación entre dos valores. Devuelve un valor booleano basado en la condición.

>	mayor que
<	menor que
==	igual a
>=	mayor o igual que
<=	menor o igual que
!=	distinto a

Lógicos

Operan con datos booleanos. Se utilizan para tomar una decisión basada en múltiples condiciones.

b and c	Devuelve True si todos los operandos son True
b or c	Devuelve True si al menos uno de los operandos es True
not b	Devuelve True si el operando b es False

De pertenencia

Identifica pertenencia en alguna secuencia (listas, strings, tuplas)

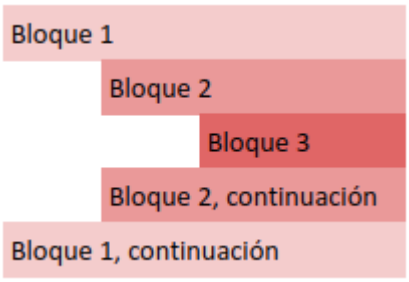
in	Devuelve True si el valor especificado se encuentra en la secuencia
not in	Devuelve True si el valor especificado no se encuentra en la secuencia

Indentación en Python

Python utiliza la indentación para delimitar estructuras, estableciendo bloques de código. No existe otra manera de finalizar las líneas, llaves ni punto y coma.

Los únicos delimitadores de los que disponemos son los dos puntos y la indentación del código. Éstos definen la lógica de nuestro progrmaa, por lo que debemos ser muy cuidadosos.

Generalmente la indentación consiste en dejar una sangría de 2 a 4 espacios desde el inicio del bloque.



Líneas comentadas

Podemos insertar líneas de comentarios en nuestro código que servirá de documentación, (y que será ignorado por el intérprete de Python) con el símbolo #. Si necesitamos insertar un bloque de comentario en lugar de sólo una línea, lo abriremos y cerraremos con una triple comilla `'''`.

```
# esta es una línea comentada

''' Este
    es un bloque
    de comentarios
'''
```

Entrada y salida de datos en Python

print()

La función print() muestra un mensaje por pantalla.

input()

La función input() espera un dato por teclado. Conviene incluir un mensaje para que el usuario sepa qué es lo que debe ingresar.

```
print("Hola") # imprime el mensaje 'Hola'

''' creamos la variable 'nombre'
    y le vamos a guardar lo que el usuario ingrese por teclado.
    Entre paréntesis le indicamos lo que el programa
    espera que se ingrese
'''
nombre = input("Por favor ingrese su nombre: ")

# Imprimimos un mensaje con el valor de la variable 'nombre'
print("Bienvenido, ", nombre, ". Gracias por probar mi código!")
```

UNIDAD III: Estructuras de control en python

Las estructuras de control nos permiten alterar el flujo de nuestro programa. Esto significa, que a lo largo del mismo, iremos planteando algunas condiciones, que van a definir qué parte del código va a ejecutarse a continuación, cuál se omitirá, y cuál se va a repetir (y cuántas veces/hasta qué momento)

Las estructuras de control pueden dividirse en 2 grandes grupos: condicionales e iterativos

- Los primeros condicionan la ejecución del bloque de código según el valor de verdad (booleano) de una expresión.
- Los últimos nos marcan la repetición de un proceso un número definido de veces o dependiendo del estado booleano de una expresión.

Pero antes debemos recordar un tema fundamental en Python: **la indentación**

Python utiliza la indentación para delimitar una estructura, permitiendo establecer bloques de código. No existen comandos para finalizar las líneas, ni llaves con las que delimitar el código. Los únicos delimitadores existentes son los dos puntos (:) y la indentación del código.

La indentación va a separar nuestros bloques de código para saber dónde comienza y dónde termina cada proceso.

Estructuras de control condicionales

Son las que definen la ejecución de un bloque de código según el valor de verdad de una expresión

Condicional if

Se utiliza para tomar decisiones. Evalúa si una expresión es True o False y ejecuta el bloque de código si el resultado es True.

Se obtiene False si se encuentra:

- un número igual a cero (0, 0.0, 0 + 0j)
- un contenedor vacío (lista, tupla, conjunto, diccionario)
- False o None

```
nro = int(input('Ingrese un número: '))
if nro < 10:
    print('El número ingresado es menor a 10')
```

else

Permite agregar un bloque de código que se ejecutará si la condición a evaluar resulta False.

```python
nro = int(input('Ingrese un número: '))
if nro < 10:
    print('El número ingresado es menor a 10')
else:
    print('El número ingresado es mayor o igual a 10')
```

elif

Es una combinación de else + if. Es decir, lo usamos si queremos evaluar otra condición si no se cumple el primer if. Podemos utilizar tantos elif como sean necesarios.

```python
nro = int(input('Ingrese un número: '))
if nro < 10:
    print('El número ingresado es menor a 10')
elif nro == 10:
    print('El número ingresado es igual a 10')
else:
    print('El número ingresado es mayor a 10')
```

Estructuras de control iterativas

Son las que permiten ejecutar un bloque de código múltiples veces (ciclos)

for

for controlado por la cantidad de elementos

En este caso, el código se va a repetir para cada uno de los elementos de la lista numeros.

```
numeros = [2, 4, 6, 23, 19, 36]
for n in numeros:
    if n % 2 == 0:
        print(n, 'es par')
    else:
        print(n, 'es impar')
```

for y la clase range

range es un tipo de dato que simula una secuencia numérica. Para construirlo necesitamos de 1 a 3 parámetros:

- range(max) crea un rango de número desde 0 hasta max-1
- range(min, max) crea un rango de número desde min hasta max-1
- range(min, max, step) crea un rango de número desde min hasta max-1, cuyos valores se van incrementando de step en step

Por ejemplo, el siguiente código va a imprimir los números pares desde el 0 hasta el 10.

```
for n in range(0,11,2):
    print(n)
```

while

Realiza múltiples iteraciones basándose en el resultado de una expresión lógica que puede tener como resultado True o False

while controlado por conteo

En este caso, el bucle while está controlado por la variable `nro`, que se va incrementando con cada ejecución.

Cuando esta variable llega a valer 10, la condición del bucle pasa a ser False y termina el ciclo.

```
suma = 0
nro = 0
while nro <=10:
    suma = nro + suma
    nro += 1
print('La suma da ' + str(suma))
```

while controlado por evento

El bucle se interrumpe cuando el usuario ingresa -1.

```
contador = 0
total = 0
nota = 99
while nota != -1:
    nota = int(input('Ingrese la nota del alumno (-1 para salir): '))
    if nota != -1:
        total += nota
        contador += 1
promedio = total / contador
print('El promedio de notas es de', promedio)
```

while con else

Una manera alternativa más legible para el código anterior se logra combinando while con else.

```
contador = 0
total = 0
nota = 99
while nota != -1:
    nota = int(input('Ingrese la nota del alumno (-1 para salir): '))
    if nota != -1:
        total += nota
        contador += 1
else:
    promedio = total / contador
    print('El promedio de notas es de', promedio)
print('-- FIN DEL PROGRAMA --')
```

Sentencias utilitarias

break

Se utiliza para interrumpir un bucle while o for.

En el siguiente ejemplo, el programa interrumpe el ciclo cuando alcanza un valor ingresado por el usuario:

```
variable = 11
parar = int(input('Dónde me detengo? [1-9]]'))
while variable > 0:
    print('Valor actual', variable)
    variable -= 1
    if variable == parar:
        break
```

También podemos interrumpir un ciclo for. En el siguiente ejemplo, el ciclo se interrumpe al encontrar un número mayor a 10 entre la lista de elementos:

```
lista = [2, 4, 5, 7, 9, 1, 3, 1000, 6, 8]
for e in lista:
    if e > 10:
        print('Nro demasiado grande')
        break
    print(e)
```

continue

La sentencia **continue** dirige el programa hacia un nuevo inicio de bucle while o for, ignorando todo lo que hay debajo, aunque no se haya terminado de ejecutar todo el proceso del ciclo.

Podemos crear un código que ignore los números pares e imprima los impares hasta el 21:

```
variable = 0
while variable < 21:
    variable += 1
    if variable % 2 == 0:
        continue
    print(variable, 'es impar')
```

En el siguiente ejemplo, se imprimen todos los números de la lista, excepto el 9.

```
lista = [2, 4, 5, 7, 9, 1, 3, 1000, 6, 8]
for e in lista:
    if e == 9:
        print('Ese número no me gusta')
        continue
    print(e)
```