Clase Nº 1

Introducción

Lenguaje Python
Versión 3 (3.x.y)

- Thonny (thonny.org)
- Python oficial (python.org)
- Anaconda Python
- Pydroid 3 (Android)

Opciones online

- https://www.onlinegdb.com/online_python_compiler
- https://www.tutorialspoint.com/execute_python3_online.php
- https://repl.it/languages/python3

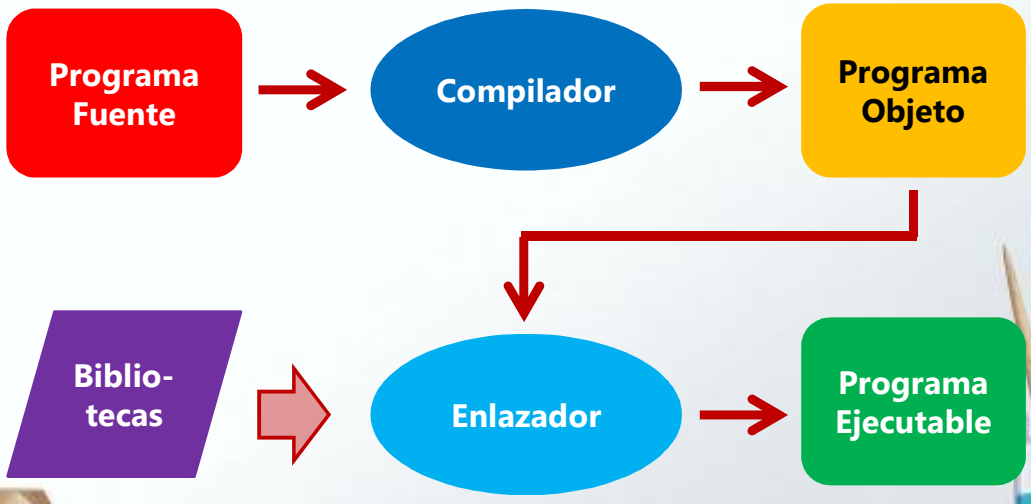
¿Qué es Python?

- Es un lenguaje de muy alto nivel.
- Es interpretado: No se compila.
- Es multiparadigma.
- El espíritu de Python ("zen") privilegia la legibilidad del código.
- Es portable.
- Es sensible a mayúsculas y minúsculas.

Compilador

- Convierte el *programa fuente*, escrito en un lenguaje de alto nivel, en un *programa objeto*.
- Este programa objeto debe ser *enlazado* o *vinculado* con bibliotecas proporcionadas por el fabricante, creando así el *programa ejecutable*.

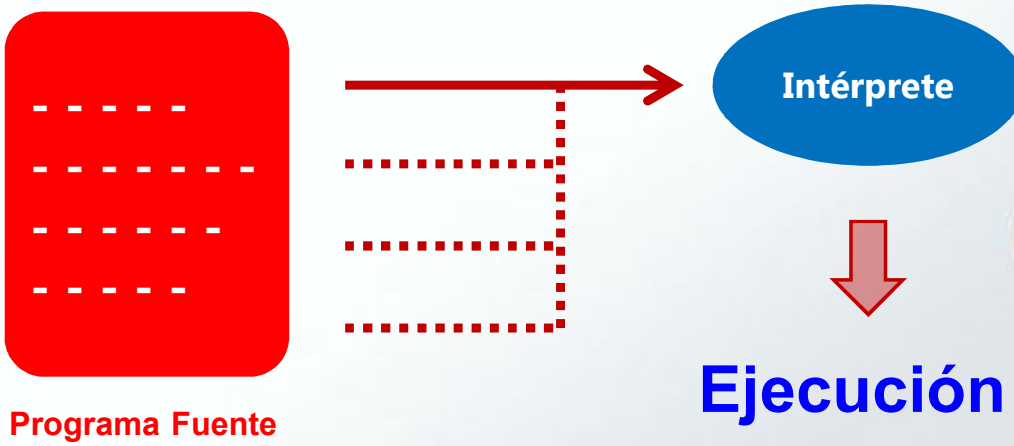
Proceso de compilación



Intérprete

- Traduce una por una las líneas del programa fuente, ejecutándolas inmediatamente.

Proceso de interpretación



¿Compilador o intérprete?

- Los programas interpretados suelen ser más lentos que los programas compilados.
- También tienen menos requisitos sintácticos que los compiladores, lo que permite probar los programas a medida que se van desarrollando.

Estructura de los programas en Python

- No tienen un formato rígido.
- Las instrucciones terminan con Enter.
- Los comentarios comienzan con el signo numeral: #

Salida por pantalla: Función print()

```
print('Hola Mundo')
print("Hola de nuevo")

print("Hola Mundo", end=" ")
print("Hola de nuevo")
```


Variables

- Se crean automáticamente con la asignación de un valor inicial. Ésto se conoce como *inicializar* la variable.
- Intentar usar una variable no inicializada provoca un error.

Variables

Reglas para crear nombres de variables:

- **Sólo letras, números y guión bajo.**
- **No pueden comenzar con un número.**
- **No pueden coincidir con las palabras reservadas del lenguaje.**

Variables

- **Los nombres de las variables *deben tener sentido*.**
- **Deben evitarse variables llamadas "l" *(ele minúscula)* u "o" *(letra o)* porque pueden confundirse fácilmente con números.**

Variables

- Pueden almacenarse números (enteros y reales), textos, etc.
- Existen valores booleanos: True y False.
- Se pueden hacer asignaciones múltiples
  - ✓ a=b=c=0
  - ✓ a, b, c = 3, "Lunes", 5.18

Variables

- Para imprimir variables y constantes se las separa con una coma:

```
dia = 5
print("Hoy es", dia, "de Marzo")
```

- Python agrega automáticamente un espacio como separador.

Variables

- **Puede lograrse mayor control de la salida impresa utilizando especificadores de conversión y el operador %:**

```
cant = 10
precio = 12.5
print("Cantidad: %6d" %cant)    # Cantidad:    10
print("Precio: %7.2f" %precio)  # Precio:   12.50
```


Variables

- **Los especificadores de conversión disponibles son:**
  - **%d para números enteros**
  - **%f para números reales**
- **Los números ubicados luego del signo de porcentaje afectan a la salida impresa.**
  - **El primero es el ancho total del número.**
  - **El segundo, la cantidad de decimales.**

Variables

- Puede escribirse un 0 delante del ancho para rellenar con ceros.

  print("%04d" %a) → *0003*

- Si hay más de una variable, éstas deben encerrarse entre paréntesis:

  print("X = %4d Y = %4d" %(x, y))

Variables

- Para ingresar valores por teclado se utiliza la función input():

  a = input("Mensaje")

- input() <u>siempre</u> devuelve un string.

Variables

- Existen funciones para convertir este string a otros tipos de dato:

```
n = int(input("Mensaje"))
r = float(input("Mensaje"))
```


Operadores aritméticos

+ Suma	- Resta
* Multiplicación	/ División real
// División entera	% Módulo o resto
** Potenciación	

Orden de evaluación

1. Potenciación
2. Menos unario
3. Multiplicación, división y módulo
4. Suma y resta

Qué resultado arroja `-2**2` ?

Asignación extendida

```
a += 1  ⇔  a = a + 1
a -= 2  ⇔  a = a - 2
a *= 3  ⇔  a = a * 3
a /= 4  ⇔  a = a / 4
a //= 5 ⇔  a = a // 5
a %= 6  ⇔  a = a % 6
a **= 7 ⇔  a = a ** 7
```

No existen operadores incrementales.

Estructuras de Control

Estructura Alternativa

Formato 1

if ***<condición>*****:**

. . . . . .

. . . . . .

. . . . . .

Estructura Alternativa

Formato 2

```
if <condición>:
    . . . . . .
else:
    . . . . . .
```


Estructura Alternativa

Formato 3

```
if <condición>:
    . . . . . .
elif <condición>:
    . . . . . .
else:
    . . . . . .
```

Estructura Alternativa

- Las condiciones y el else van seguidas del carácter “dos puntos”.
- La sangría o indentación es lo que establece el alcance del if.
- Python recomienda una sangría standard de 4 espacios, sin tabs.
- La sangría debe ser uniforme.

Operadores relacionales

== igual

> mayor

< menor

>= mayor o igual

<= menor o igual

!= distinto

Operadores lógicos

and

or

not

Condiciones encadenadas

if a<b>c:

equivale a

if a<b and b>c:

Condiciones encadenadas

¡Cuidado con las condiciones encadenadas!

if n >= 10 <= 100:

equivale a

if n >= 10 and 10 <= 100:

La variable debe escribirse en el medio:

if 10 <= n <= 100:

Operador condicional

<var> = <valor1> if <condición> else <valor2>

Ejemplo:

a = b if b>=0 else -b

Ejemplo 1

Ingresar un número entero e imprimir el nombre del mes correspondiente (1: Enero, 2: Febrero, 3: Marzo...)


```
mes = int(input("Mes ? "))
if mes == 1:
    print("Enero")
elif mes == 2:
    print("Febrero")
elif mes == 3:
    print("Marzo")
[ . . . ]
elif mes == 12:
    print("Diciembre")
else:
    print("Mes inválido")
```

Estructura Iterativa

Instrucción while

while *<condición>*:

. . . . . .

. . . . . .

. . . . . .

Estructura Iterativa

- **La condición va seguida del carácter “dos puntos”.**
- **La sangría o indentación es lo que establece el alcance del ciclo.**
- **Python recomienda una sangría standard de 4 espacios, sin tabs.**
- **La sangría debe ser uniforme.**

Estructura Iterativa

- **Instrucción break:** Abandona el último ciclo ejecutado.
- **Cláusula else:** El código de esta cláusula sólo se ejecuta si el ciclo terminó en forma normal, es decir sin break.

Ejemplo 2

Uso de while, break y else:

Leer un número entero e imprimir un mensaje indicando si se trata de un número primo o no.

```
n = int(input("Ingrese un número: "))
divisor = 2
while divisor<n:
    if n % divisor == 0:
        print(n, "no es un número primo")
        break
    divisor = divisor + 1
else:
    print(n, "es un número primo")
```


Estructura Iterativa

- **Instrucción continue:** Fuerza una nueva iteración sin haber llegado al final de la anterior.

Ejemplo 3

Uso de while y continue

Imprimir los valores de 1/x entre -3 y 3.


```
x = -4
while x<3:
    x = x + 1
    if x == 0:
        continue
    print("%2d: %5.2f" %(x,1/x))
```

Estructura Iterativa

Instrucción for

for <*variable*> in <*secuencia*>:

. . . . . .

. . . . . .

. . . . . .

Estructura Iterativa

- **for se utiliza para recorrer un *iterable*.**
- **Este iterable puede ser un rango, una lista, una cadena, un archivo, una tupla, un conjunto o un diccionario.**
- **La cláusula else y las instrucciones break y continue hacen lo mismo que en while.**

Función range()

- Genera una secuencia de números enteros.
- Admite tres formas de utilización.

Función range()

Formato 1:

range(<vfinal>)

- Genera una secuencia de números enteros entre 0 y <vfinal>.
- <vfinal> no está incluido.

Función range()

Formato 2:

range(<vinicial>, <vfinal>)

- **Genera una secuencia de enteros entre <vinicial> y <vfinal>.**
- **<vfinal> no está incluido.**

Función range()

Formato 3:

range(<vinicial>, <vfinal>, <inc>)

- **Genera una secuencia de enteros entre <vinicial> y <vfinal> con incremento <inc>.**
- **<vfinal> no está incluido.**

Función range()

Formato 3:

range(<vinicial>, <vfinal>, <inc>)

- **El incremento puede ser negativo.**
- **En este caso el valor inicial deberá ser mayor que el valor final.**

Ejemplo 4

Uso de for y range()

Imprimir los números impares entre 1 y N.

```
# Imprimir los números impares entre 1 y N
n = int(input("Ingrese un número: "))
for i in range(1, n+1, 2):
    print(i, end=" ")
print()
```


Ejercitación

- **Buscar en Internet el *Zen de Python* y analizarlo.**
- **Escribir un programa para resolver el antiguo acertijo chino que se describe a continuación:**

  "He contado 35 cabezas y 94 patas entre las gallinas y los conejos de mi granja. ¿Cuántos conejos y cuántas gallinas tengo?"

  Sugerencia: Verificar todas las combinaciones posibles hasta hallar la correcta; no realizar una resolución algebraica.