

Clase N° 1

Introducción

© Lic. Ricardo Thompson

Lenguaje Python Versión 3 (3.x.y)

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

© Lic. Ricardo Thompson

Opciones online

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

© Lic. Ricardo Thompson

¿Qué es Python?

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

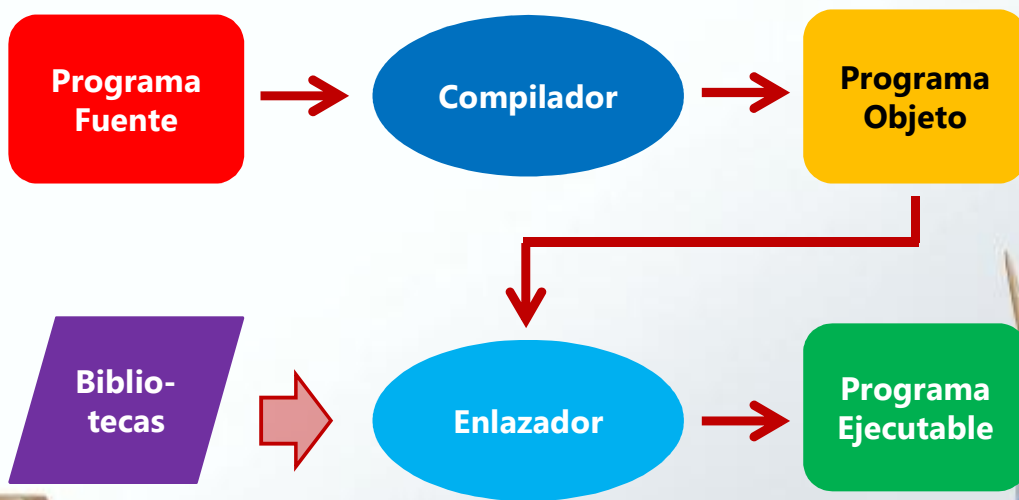
© Lic. Ricardo Thompson

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*.

© Lic. Ricardo Thompson

Proceso de compilación



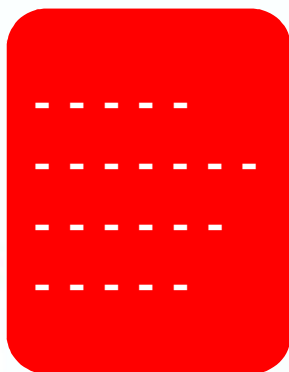
© Lic. Ricardo Thompson

Intérprete

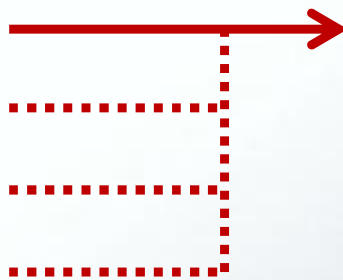
- Traduce una por una las líneas del programa fuente, ejecutándolas inmediatamente.

© Lic. Ricardo Thompson

Proceso de interpretación



Programa Fuente



Intérprete



Ejecución

© Lic. Ricardo Thompson

¿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.

© Lic. Ricardo Thompson

Estructura de los programas en Python

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

© Lic. Ricardo Thompson

Salida por pantalla: Función print()

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

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

© Lic. Ricardo Thompson

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.

© Lic. Ricardo Thompson

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.

© Lic. Ricardo Thompson

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.

© Lic. Ricardo Thompson

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`

© Lic. Ricardo Thompson

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.

© Lic. Ricardo Thompson

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
```

© Lic. Ricardo Thompson

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.

© Lic. Ricardo Thompson

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))
```

© Lic. Ricardo Thompson

Variables

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

```
a = input("Mensaje")
```

- `input()` siempre devuelve un string.

© Lic. Ricardo Thompson

Variables

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

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

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

© Lic. Ricardo Thompson

Operadores aritméticos

+ Suma

- Resta

*** Multiplicación**

/ División real

// División entera

% Módulo o resto

**** Potenciación**

© Lic. Ricardo Thompson

Orden de evaluación

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

Qué resultado arroja **$-2^{**}2$** ?

© Lic. Ricardo Thompson

Asignación extendida

$a += 1 \Leftrightarrow a = a + 1$

$a -= 2 \Leftrightarrow a = a - 2$

$a *= 3 \Leftrightarrow a = a * 3$

$a /= 4 \Leftrightarrow a = a / 4$

$a //= 5 \Leftrightarrow a = a // 5$

$a \% = 6 \Leftrightarrow a = a \% 6$

$a ** = 7 \Leftrightarrow a = a ** 7$

No existen operadores incrementales.

© Lic. Ricardo Thompson

Estructuras de Control

© Lic. Ricardo Thompson

Estructura Alternativa

Formato 1

if *<condición>*:

.....

.....

.....

© Lic. Ricardo Thompson

Estructura Alternativa

Formato 2

if *<condición>*:

.....

else:

.....

© Lic. Ricardo Thompson

Estructura Alternativa

Formato 3

if *<condición>*:

.....

elif *<condición>*:

.....

else:

.....

© Lic. Ricardo Thompson

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.

© Lic. Ricardo Thompson

Operadores relacionales

== igual

> mayor

< menor

>= mayor o igual

<= menor o igual

!= distinto

© Lic. Ricardo Thompson

Operadores lógicos

and
or
not

© Lic. Ricardo Thompson

Condiciones encadenadas

if $a < b > c$:

equivale a

if $a < b$ and $b > c$:

© Lic. Ricardo Thompson

Condiciones encadenadas

¡Cuidado con las condiciones encadenadas!

if $n \geq 10 \leq 100$:

equivale a

if $n \geq 10$ and $10 \leq 100$:

La variable debe escribirse en el medio:

if $10 \leq n \leq 100$:

© Lic. Ricardo Thompson

Operador condicional

$\langle \text{var} \rangle = \langle \text{valor1} \rangle$ if $\langle \text{condición} \rangle$ else $\langle \text{valor2} \rangle$

Ejemplo:

$a = b$ if $b \geq 0$ else $-b$

© Lic. Ricardo Thompson

Ejemplo 1

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

© Lic. Ricardo Thompson

```
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")
```

© Lic. Ricardo Thompson

Estructura Iterativa

Instrucción while

while *<condición>*:

.....

.....

.....

© Lic. Ricardo Thompson

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.

© Lic. Ricardo Thompson

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.

© Lic. Ricardo Thompson

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.

© Lic. Ricardo Thompson

```
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")
```

© Lic. Ricardo Thompson

Estructura Iterativa

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

© Lic. Ricardo Thompson

Ejemplo 3

Uso de while y continue

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

© Lic. Ricardo Thompson

```
x = -4
```

```
while x < 3:
```

```
    x = x + 1
```

```
    if x == 0:
```

```
        continue
```

```
    print("%2d: %5.2f" % (x, 1/x))
```

© Lic. Ricardo Thompson

Estructura Iterativa

Instrucción for

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

.....
.....
.....

© Lic. Ricardo Thompson

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**.

© Lic. Ricardo Thompson

Función range()

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

© Lic. Ricardo Thompson

Función range()

Formato 1:

range(<vfinal>)

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

© Lic. Ricardo Thompson

Función range()

Formato 2:

range(<vinicial>, <vfinal>)

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

© Lic. Ricardo Thompson

Función range()

Formato 3:

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

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

© Lic. Ricardo Thompson

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.

© Lic. Ricardo Thompson

Ejemplo 4

Uso de for y range()

**Imprimir los números impares
entre 1 y N.**

© Lic. Ricardo Thompson

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()
```

© Lic. Ricardo Thompson

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.

© Lic. Ricardo Thompson