



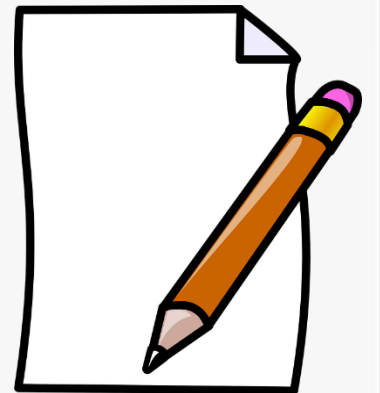
TI2041

PROGRAMACIÓN BACK END

Unidad 1: Tecnología del Lado del Servidor
Clase 2

APRENDIZAJE ESPERADO

Realiza un programa del lado del servidor, de acuerdo a la sintaxis del lenguaje.



MOTIVACIÓN



+



EN ESTA CLASE

- Recordar programación en Python:
 - Características
 - Variables y Expresiones
 - Instrucciones Básicas
 - Estándar de Programación
- Django Framework:
 - ¿Qué es Django?
 - Cómo Instalar Django
 - Cómo Crear un Proyecto
 - Estructura de un Proyecto
 - Ejemplo Práctico

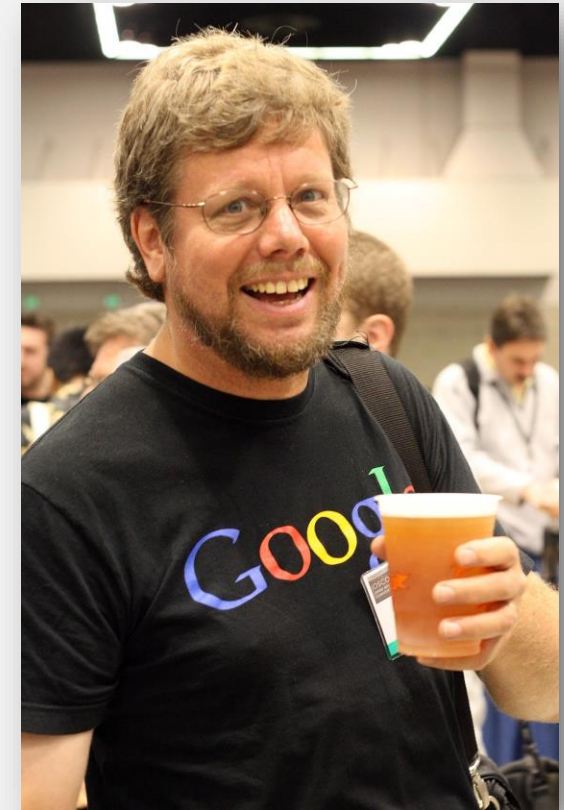


CARACTERÍSTICAS DEL LENGUAJE

- Python es un **lenguaje de alto nivel interpretado**, cuya filosofía hace hincapié en la legibilidad de su código.
- Python es:
 - **De propósito general**, por lo que es útil para realizar múltiples tareas y desarrollar distintos tipos de software, y no está orientado a un conjunto de problemas en específico.
 - **Multiparadigma**, es decir, permite trabajar sobre distintos enfoques o paradigmas de programación.
 - **Interpretado**, lo que implica que el código en Python se transforma en código de máquina cuando se necesita la instrucción en específico.

CARACTERÍSTICAS DEL LENGUAJE

- A pesar de que pueda parecer algo muy nuevo, Python remonta su origen **a principios de los años 90**, cuando Guido Van Rossum, un trabajador del Centrum Wiskunde & Informatica (CWI), un centro de investigación holandés, tuvo la idea de desarrollar un nuevo lenguaje basándose en un proyecto anterior, el lenguaje de programación “ABC”, que él mismo había desarrollado junto a sus compañeros.
- Su filosofía fue la misma desde el primer momento: **crear un lenguaje de programación que fuera muy fácil de aprender, escribir y entender**, sin que esto frenara su potencial para crear cualquier tipo de aplicación.



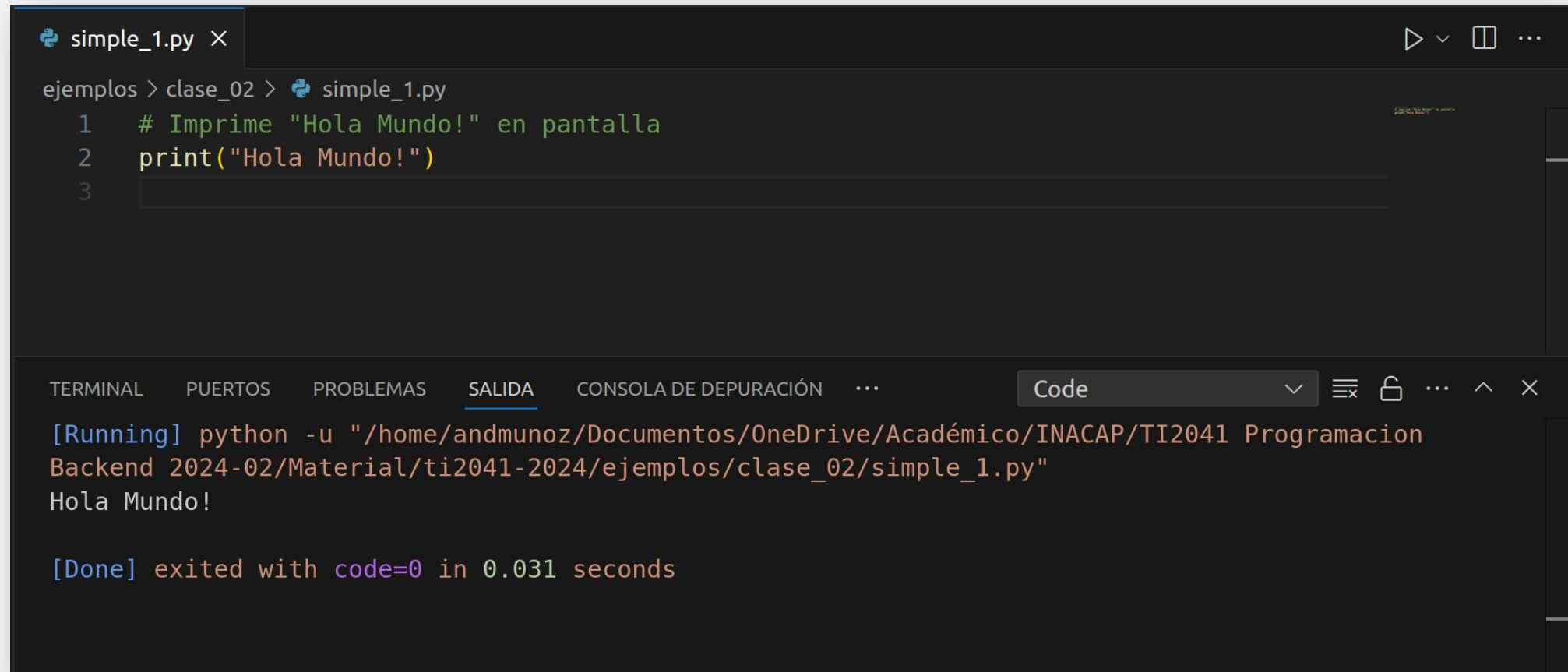
CARACTERÍSTICAS DEL LENGUAJE

- Su **versatilidad**, lo hace ideal en muchos ámbitos como:
 - Data analytics y big data
 - Data mining
 - Data science
 - Inteligencia artificial
 - Blockchain
 - Machine learning
 - Desarrollo web
 - Juegos y gráficos 3D
 - Entre otros...



CARACTERÍSTICAS DEL LENGUAJE

- Ejemplo de código Python:



The screenshot shows a code editor window with a file named `simple_1.py`. The code contains a single line of Python code: `print("Hola Mundo!")`. Below the code editor, the output of running the script is displayed. The output shows the command `python -u "/home/andmunoz/Documentos/OneDrive/Académico/INACAP/TI2041 Programacion Backend 2024-02/Material/ti2041-2024/ejemplos/clase_02/simple_1.py"` being executed, followed by the output `Hola Mundo!` and a status message `[Done] exited with code=0 in 0.031 seconds`.

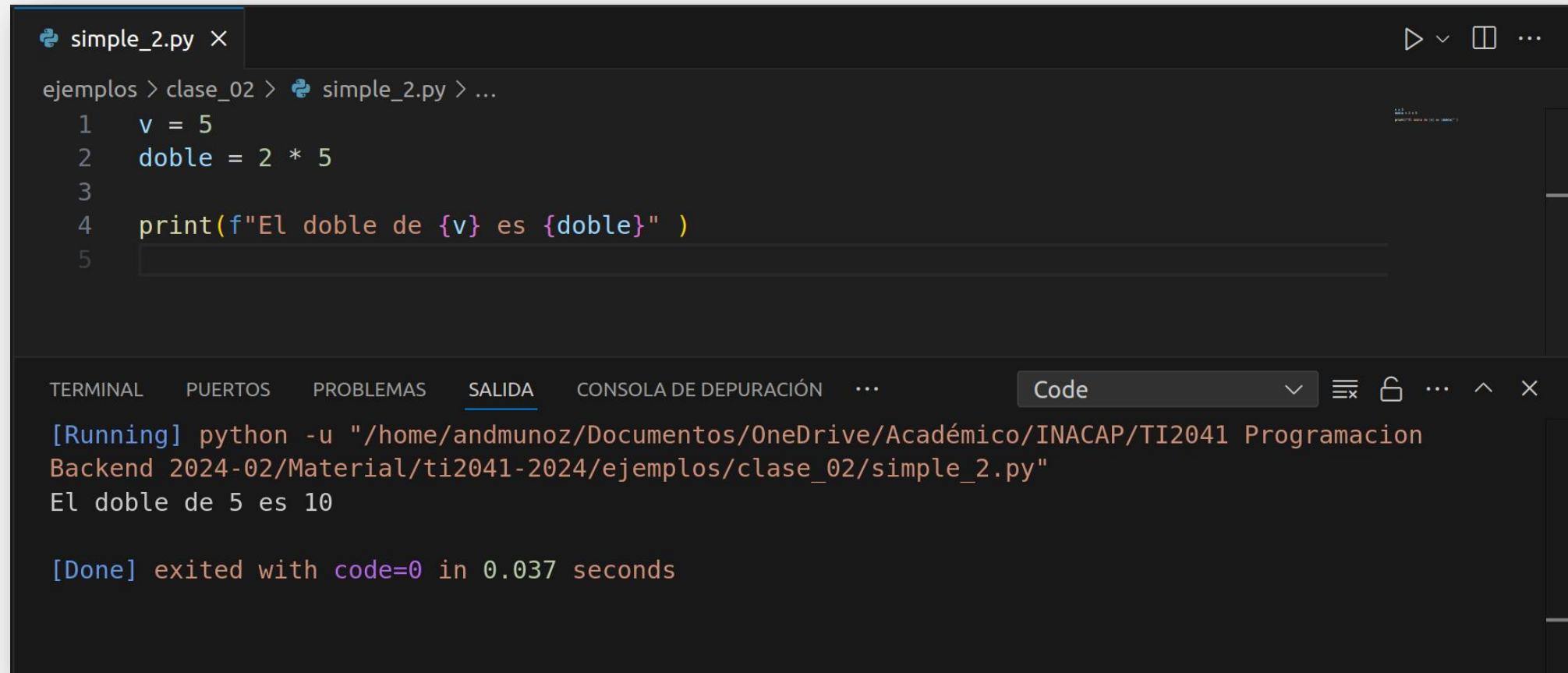
```
simple_1.py X
ejemplos > clase_02 > simple_1.py
1 # Imprime "Hola Mundo!" en pantalla
2 print("Hola Mundo!")
3

[Running] python -u "/home/andmunoz/Documentos/OneDrive/Académico/INACAP/TI2041 Programacion Backend 2024-02/Material/ti2041-2024/ejemplos/clase_02/simple_1.py"
Hola Mundo!

[Done] exited with code=0 in 0.031 seconds
```


CARACTERÍSTICAS DEL LENGUAJE

- Ejemplo de código Python:



The screenshot shows a code editor window with a file named `simple_2.py`. The code defines a variable `v` with the value 5, calculates `doble = 2 * 5`, and prints the result using an f-string. Below the code editor, the output console shows the command used to run the script and the resulting output: "El doble de 5 es 10".

```
simple_2.py x
ejemplos > clase_02 > simple_2.py > ...
1  v = 5
2  doble = 2 * 5
3
4  print(f"El doble de {v} es {doble}" )
5

[Running] python -u "/home/andmunoz/Documentos/OneDrive/Académico/INACAP/TI2041 Programacion Backend 2024-02/Material/ti2041-2024/ejemplos/clase_02/simple_2.py"
El doble de 5 es 10

[Done] exited with code=0 in 0.037 seconds
```

VARIABLES Y EXPRESIONES

- En Python, al igual que en la mayoría de los lenguajes de programación populares, existe un operador especial, llamado **asignación**.
- Este operador permite **almacenar un valor o el resultado de una expresión** con un nombre determinado por el programador para usarlo posteriormente
- La **asignación** tiene la siguiente estructura:

<identificador> = <expresión>

VARIABLES Y EXPRESIONES

- Por ejemplo, consideremos la siguiente expresión:

`resultado = 2 ** 8`

- Python primero evalúa la expresión, obteniendo el **valor 256** y luego lo almacena en una variable llamada **resultado**.
- Podemos imaginar una variable como una **caja en la memoria** del programa en la cuál el valor está guardado



VARIABLES Y EXPRESIONES

- Una asignación es una sentencia con la siguiente estructura:

<identificador> = <expresión>

- El primer carácter no puede ser un dígito
- Puede llevar letras, dígitos y el carácter subrayado (`_`)
- No puede coincidir con las palabras reservadas de Python:
 - `and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, raise, return, try, while, yield`
- Un valor constante (un número o texto)
- Una operación entre números
- Una operación entre variables y constantes previamente declaradas
- Mezcla entre operaciones, variables y números

VARIABLES Y EXPRESIONES

- En Python podemos almacenar valores como **variables** o **constantes**.
- En la práctica, diferenciamos las constantes de las variables, para indicarle a otros programadores **qué valores podrían ser modificados y qué valores no**.
- Por ejemplo, si hacemos un programa que calcule el área y perímetro de una circunferencia necesitamos definir dos valores iniciales:
 - El **radio**, que correspondería a una variable, pues su valor puede modificarse
 - El valor de **π** que corresponde a una constante, pues este nunca cambia

VARIABLES Y EXPRESIONES

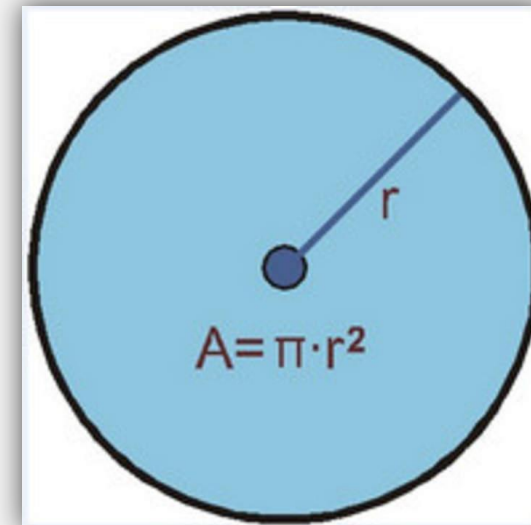
- En este caso los nombres perímetro, area y radio corresponderían a variables, pues son valores que dependen de otros o que podrían variar dependiendo del caso.
- En cambio PI es un valor matemático definido y siempre debería ser igual.

$$PI = 3.1415926535897931$$

$$\text{radio} = 2$$

$$\text{perimetro} = 2 * PI * \text{radio}$$

$$\text{area} = PI * \text{radio} ** 2$$

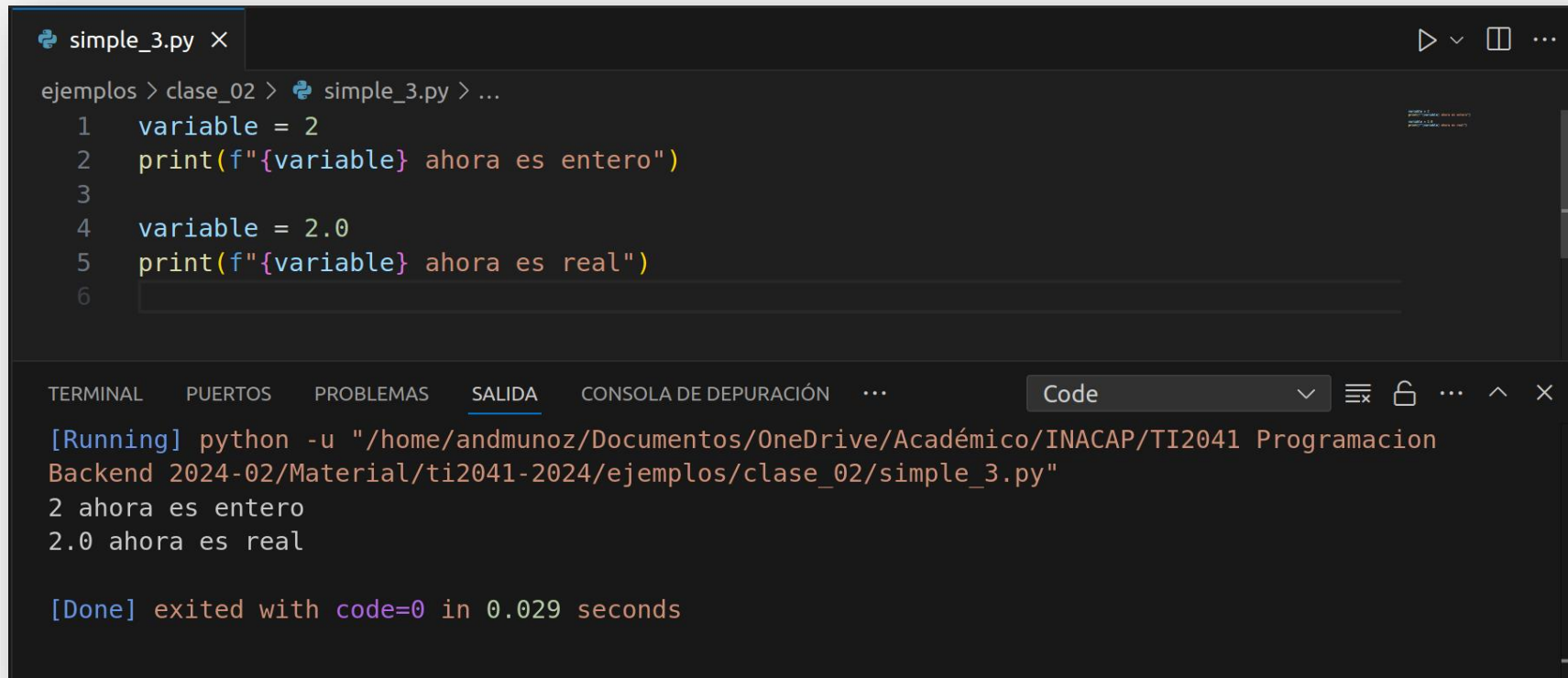


VARIABLES Y EXPRESIONES

- En Python se utilizan reglas distintas para variables y constantes:
 - Las variables se escriben en con guiones bajos en minúsculas y separando palabras distintas (**lower_case_with_underscores**).
 - Buenos nombres de variables serían valor, color, palabra_original, lista_estudiantes.
 - Las constantes en cambio se escriben con mayúsculas y también separando las variables con guiones bajos (**UPPER_CASE_WITH_UNDERSCORES**).
 - Buenos nombres para constantes serían VALOR_PI, NOMBRE_ARCHIVO, GRAVEDAD, ARCHIVO_SALIDA

VARIABLES Y EXPRESIONES

- Python es un **lenguaje no tipado**, que significa que las variables no tienen un tipo específico de valor al momento de ser utilizadas.



```
simple_3.py X
ejemplos > clase_02 > simple_3.py > ...
1 variable = 2
2 print(f"{variable} ahora es entero")
3
4 variable = 2.0
5 print(f"{variable} ahora es real")
6

[Running] python -u "/home/andmunoz/Documentos/OneDrive/Académico/INACAP/TI2041 Programacion Backend 2024-02/Material/ti2041-2024/ejemplos/clase_02/simple_3.py"
2 ahora es entero
2.0 ahora es real

[Done] exited with code=0 in 0.029 seconds
```

OPERADORES

- Los tipos de valores que puede almacenar una **variable numérica** en Python son básicamente 3:

Tipo	Valor Esperado
int	Número entero (sin decimales)
float	Número real (con decimales)
bool	Binario (1 o 0)

Nota: Los valores binarios no se operan como números en Python, ya veremos por qué.

OPERADORES

- Operadores Numéricos:

Operador	Significado	Ejemplo	Resultado	Precedencia
+	Suma	$7 + 2$	9	3
-	Resta	$7 - 2$	5	3
*	Multiplicación	$7 * 2$	14	2
/	División	$7 / 2$	3.5	2
%	Módulo	$7 \% 2$	1	2
//	Cociente	$7 // 2$	3	2
**	Potencia	$7 ** 2$	49	1

OPERADORES

- En particular, cuando hablamos del tipo `bool` (binario), Python tiene una representación con unas constantes especiales: **True** y **False**.
- A estos valores se les conoce como "valores de verdad" porque se pueden evaluar a partir de una **comparación lógica**.
- Las más conocidas de las operaciones booleanas corresponden a los **operadores de comparación**.

OPERADORES

- Operadores de Comparación:

Operador	Significado	Ejemplo	Resultado
>	Mayor	4.3 > 3.2	True
>=	Mayor o igual	4.0 >= 4.1	False
<	Menor	-2 < 0	True
<=	Menor o igual	-3.14 <= -3.2	False
==	Igual	2.0 == 2	True
!=	Distinto (no igual)	-2 != -2.1	True

OPERADORES

- Además existen **operadores lógicos** que se utilizan para componer expresiones booleanas.
- Estos operadores no son propios de Python, pues son comunes en el álgebra:
 - La **conjunción** o “y lógico” (**and**), que resulta verdadero si y solo si **todas** las sub-expresiones son verdaderas:

$$x > y \text{ and } y \leq z$$

- La **disyunción** u “o lógico” (**or**), que resulta verdadero si **al menos una** de las sub-expresiones es verdadera:

$$x \neq y \text{ or } x \leq z \text{ or } y < z$$

OPERADORES

- Finalmente existe el operador **negación** (**not**) para **invertir** el valor de verdad:

not 50 > 4 → **not** True → False

not -40.4 >= 44.5 → **not** False → True

- Existen otros operadores que entregan resultados booleanos que utilizaremos, pero con éstos es más que suficiente (por ahora).

OPERADORES

- Los operadores booleanos también tienen **reglas de precedencia**:
 - Los operadores de comparación tienen **menor precedencia** que los operadores aritméticos.
 - Luego tiene precedencia el **operador de negación** (**not**), luego las **conjunciones** (**and**) y finalmente las **disyunciones** (**or**).
- Podemos alterar esta precedencia usando paréntesis:

$x + 7 > y \text{ and } \text{not } y ** 2 + x \leq z \text{ or } x + z \neq y$

es equivalente a

$((x + 7) > y) \text{ and } (\text{not } ((y ** 2) + x \leq z)) \text{ or } ((x + z) \neq y)$

SENTENCIAS

- La **sentencia if** permite condicionar la ejecución de un bloque de sentencias al **cumplimiento de una condición**.
 - La condición debe resultar siempre en **un booleano**.
 - El bloque condicionado debe estar **indentado**.
 - Para volver al flujo no condicionado es necesario volver al **nivel previo de indentación**.

if <condición> :

Se ejecuta si la condición se cumple

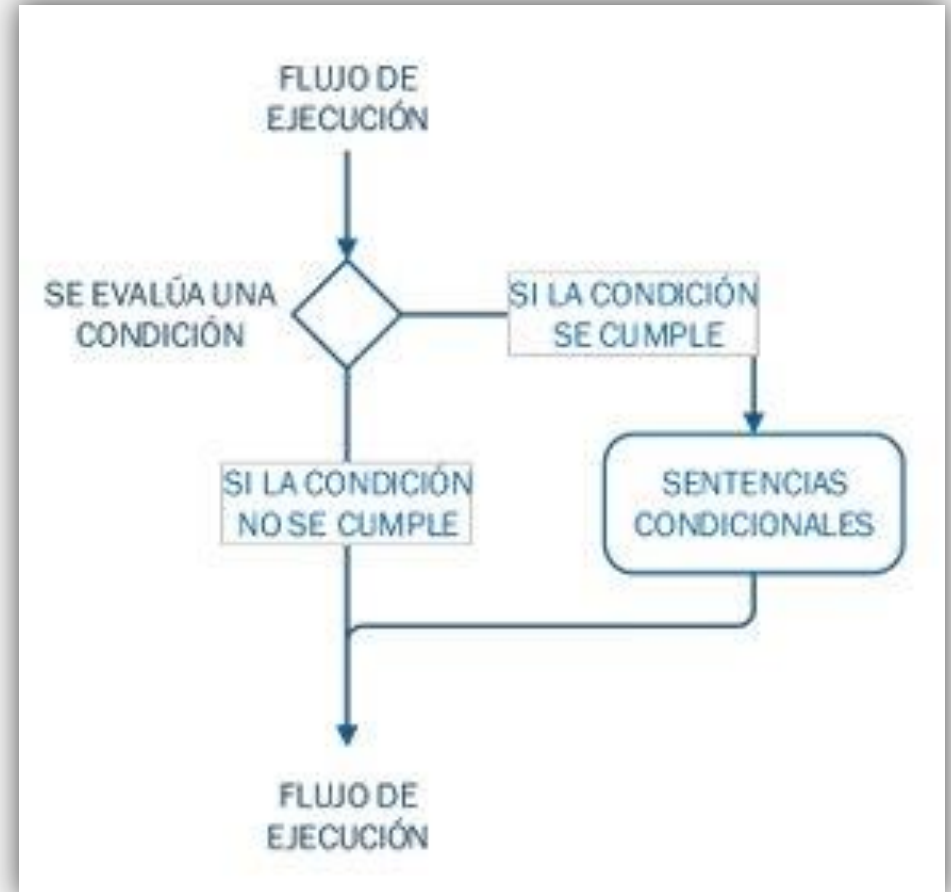
<Bloque de sentencias condicionales>

Se ejecuta si la condición se cumplió o no

<Bloque de sentencias que sigue>

SENTENCIAS

- El **flujo de ejecución** (como que fuera flujo de agua en tuberías) pasa de largo si la condición no se cumple, es decir, resulta con **valor falso**.
- Pero **ejecuta las sentencias condicionales** si la condición se cumple, es decir, resulta con **valor verdadero**.



SENTENCIAS

- Ejemplo:

```
simple_4.py X
ejemplos > clase_02 > simple_4.py > ...
1  valor = int(input("Ingrese un valor? "))
2  resultado = 2
3  if valor > 0 :
4      resultado = resultado + valor
5      print("El valor es positivo")
6  print(f"El resultado es {resultado}")
7
```

¿Qué pasa si ingresamos el valor 10, -4 o 0? ¿Cuál sería el resultado obtenido?

SENTENCIAS

- La **sentencia if-else** permite dividir el flujo de la ejecución de dos bloques de sentencias según el **cumplimiento de una condición**.

```
if <condición> :
```

```
    # Se ejecuta si la condición se cumple
```

```
    <Bloque de sentencias condicionales>
```

```
else :
```

```
    # Se ejecuta si la condición no se cumple
```

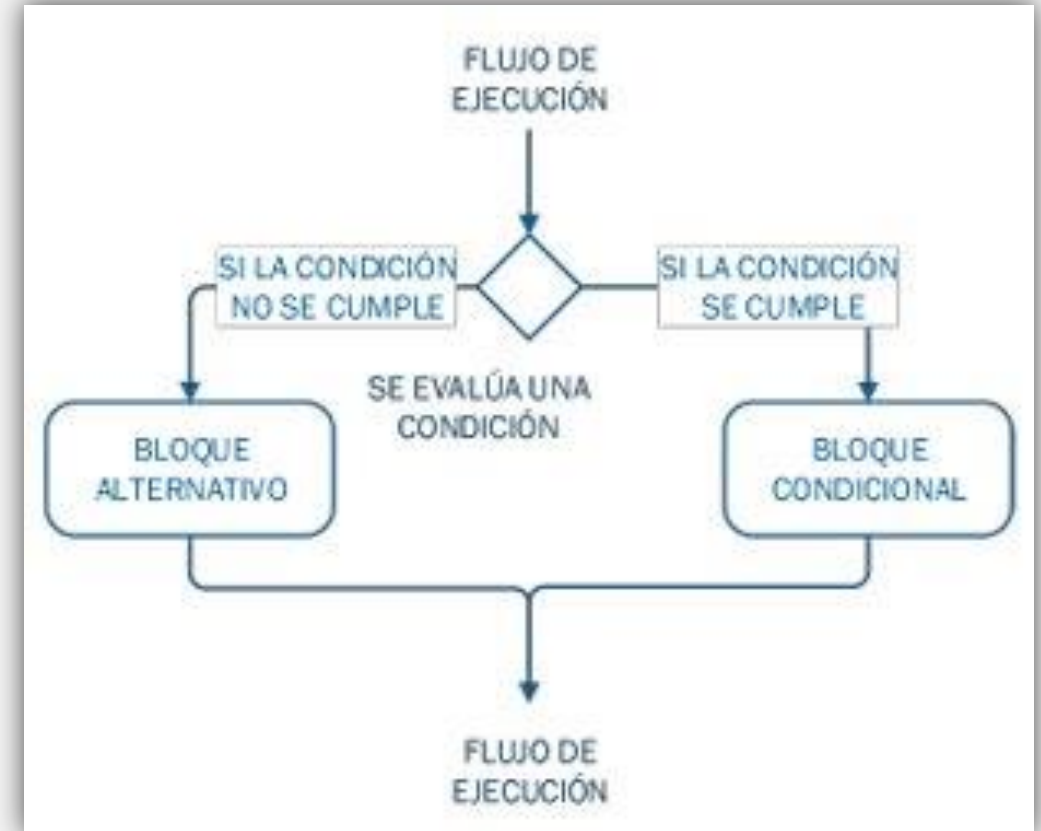
```
    <Bloque de sentencias alternativas>
```

```
# Se ejecuta si la condición se cumplió o no
```

```
<Bloque de sentencias que sigue>
```

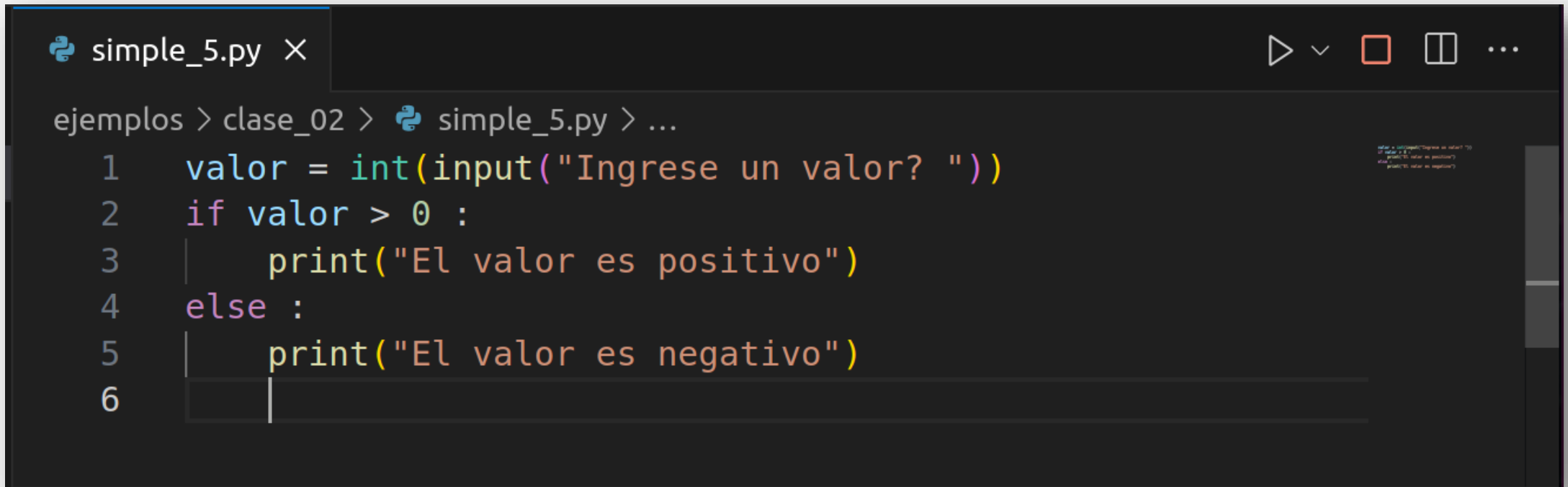
SENTENCIAS

- El **flujo de ejecución** pasa por el **bloque condicional** si la condición se cumple, es decir, si resulta verdadero.
- En caso de que la condición no se cumpla, es decir, resulta falso, pasa por el **bloque alternativo**.



SENTENCIAS

- Ejemplo:



```
simple_5.py X
ejemplos > clase_02 > simple_5.py > ...
1 valor = int(input("Ingrese un valor? "))
2 if valor > 0 :
3     print("El valor es positivo")
4 else :
5     print("El valor es negativo")
6
```

SENTENCIAS

- La **sentencia if-elif-else** permite dividir el flujo de la ejecución de tres o más bloques de sentencias según el **cumplimiento de dos o más condiciones**.

```
if <condición 1> :  
    # Se ejecuta si la condición 1 se cumple  
    <Bloque de sentencias condicionales 1>  
elif <condición 2> :  
    # Se ejecuta si la condición 1 no se cumple  
    # y la condición 2 se cumple  
    <Bloque de sentencias condicionales 2>  
...  
else :  
    # Se ejecuta si las condiciones no se cumplen  
    <Bloque de sentencias alternativas>  
# Se ejecuta si las condiciones se cumplieron o no  
<Bloque de sentencias que sigue>
```

SENTENCIAS

- El **flujo de ejecución** pasa por el **primer bloque condicional** si su condición se cumple, es decir, si resulta verdadero.
- Si no, pasa por el **segundo bloque condicional** si su condición respectiva se cumple.
- En caso de que ninguna condición se cumpla, es decir, todas resulten falsas, pasa por el **bloque alternativo**.



SENTENCIAS

- Ejemplo:

```
simple_6.py X
```

```
ejemplos > clase_02 > simple_6.py > ...
```

```
1 valor = int(input("Ingrese un valor? "))
2 if valor % 2 == 0 :
3     print(f"{valor} es divisible por 2")
4 elif valor % 3 == 0 :
5     print(f"{valor} es divisible por 3")
6 else :
7     print(f"{valor} es divisible por 5")
```

SENTENCIAS

- Una **Iteración o Bucle** es una estructura que nos permite **repetir una serie de instrucciones** en base a una lógica en particular.
- Esto permite realizar **operaciones repetitivas** que van orientadas a obtener un resultado o realizar acciones acumulativas en nuestros algoritmos.
- Existen varias formas de hacer iteraciones, pero veremos las sentencias básicas que son **while** y **for**.

While <condición> :

Se ejecuta si la condición se cumple

<Bloque de sentencias a repetir>

Se ejecuta después de terminado el ciclo

<Bloque de sentencias que sigue>

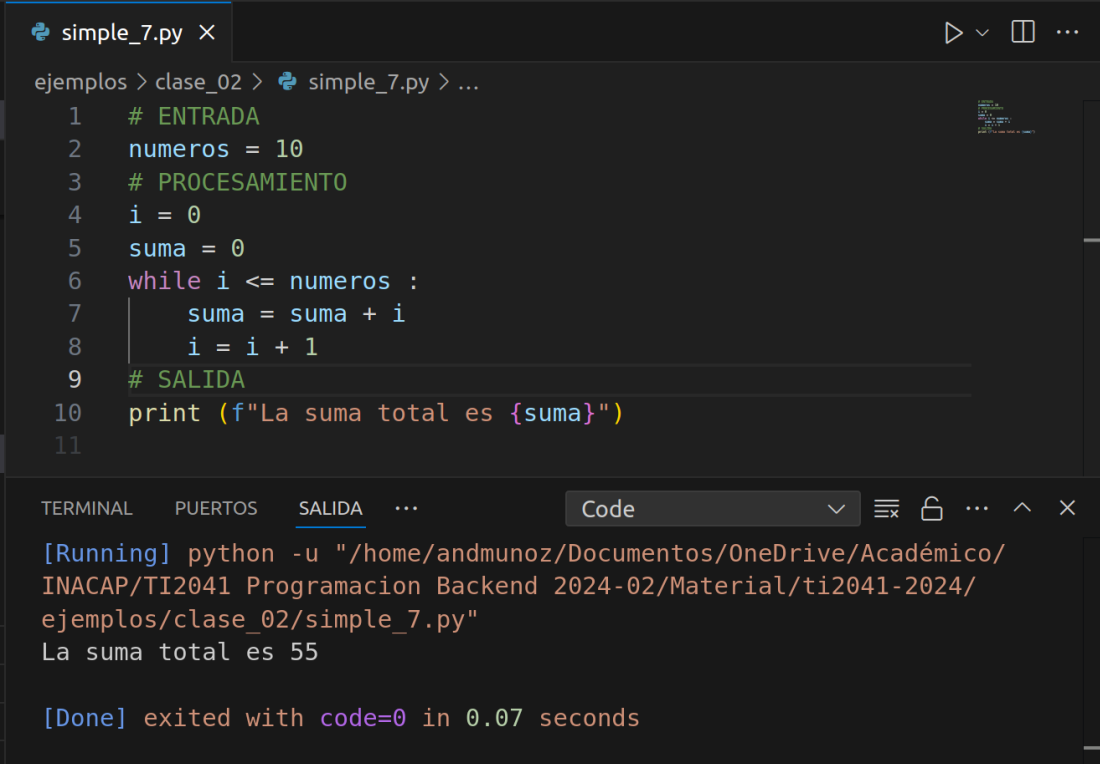
SENTENCIAS

- A diferencia de un **if**, cuando un bloque de sentencias condicionado por un **while** alcanza su fin **la condición se vuelve a evaluar**.
- Esto implica que el código **puede repetirse**.
- Pero en algún momento se debe encontrar **una forma de salir** del **while**.



SENTENCIAS

- Por ejemplo, si se quieren sumar los 10 primeros números naturales, se puede resolver de 3 formas diferentes:
 - **De manera explícita**, sumando los números en una misma variable.
 - **Con la fórmula de la telescópica**, utilizando la fórmula para sumar los n primeros términos de una sumatoria
 - A través de **un ciclo**.
- La tercera solución es la que nos interesa.



```
simple_7.py x
ejemplos > clase_02 > simple_7.py > ...
1  # ENTRADA
2  numeros = 10
3  # PROCESAMIENTO
4  i = 0
5  suma = 0
6  while i <= numeros :
7      suma = suma + i
8      i = i + 1
9  # SALIDA
10 print (f"La suma total es {suma}")
11

TERMINAL  PUERTOS  SALIDA  ...  Code
[Running] python -u "/home/andmunoz/Documentos/OneDrive/Académico/INACAP/TI2041 Programacion Backend 2024-02/Material/ti2041-2024/ejemplos/clase_02/simple_7.py"
La suma total es 55

[Done] exited with code=0 in 0.07 seconds
```


ERRORES Y EXCEPCIONES

- Al construir una solución programática, es muy común que uno **cometa errores**.
- En todo lenguaje de programación, los errores se pueden clasificar en 2 tipos: de **Sintaxis** y en **Tiempo de Ejecución**.
 - Los **errores de sintaxis** son aquellos que **impiden la ejecución del programa**, porque el intérprete no lo puede entender.
 - Los **errores en tiempo de ejecución** son aquellos que, a pesar de que el programa se ejecuta, provocan que **se detenga la ejecución** antes de lo esperado.
- Estos últimos, también son conocidos como **Excepciones**.

ERRORES Y EXCEPCIONES

- Las **Excepciones** muchas veces ocurren por datos que no podemos evitar que ocurran.
- Por ejemplo, cuando hacemos una división, y el usuario o un cálculo da que el divisor es 0, eso provoca la excepción **ZeroDivisionError**.
- Esto es posible de evitar que ocurre utilizando bloques **if** o **while** que permitan **validar antes** de que se ejecute el cálculo que provoca la excepción.
- Sin embargo, muchas veces es ineficiente realizar **flujos alternativos por cada excepción**.

ERRORES Y EXCEPCIONES

- Con **try-except**, se puede validar casos excepcionales utilizando una **sección crítica**.
- La sección crítica es un conjunto de instrucciones que conocemos a priori y **pueden provocar una excepción** en tiempo de ejecución.

try:

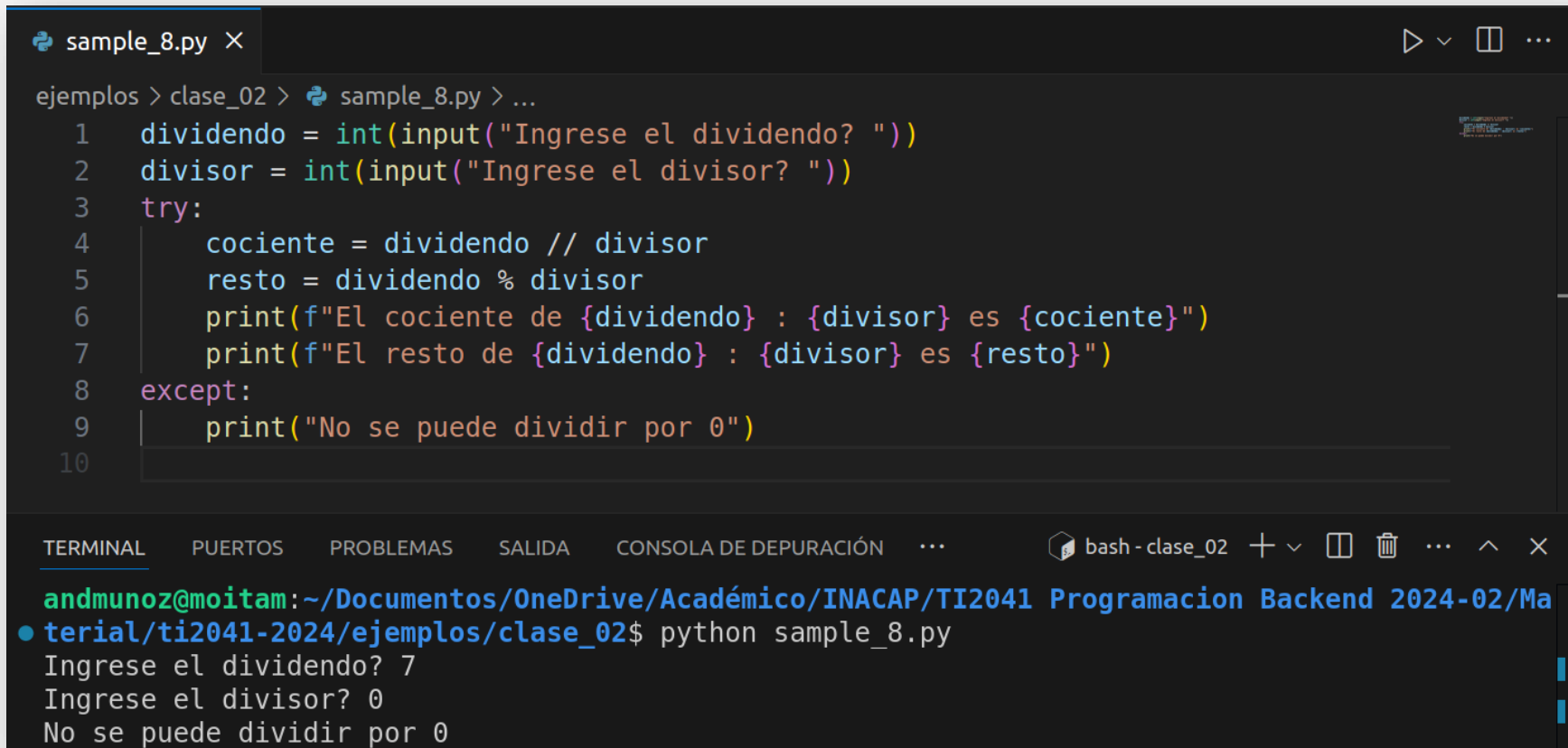
Bloque que puede provocar una excepción

except:

Bloque que se ejecuta cuando ocurre una excepción

ERRORES Y EXCEPCIONES

- Ejemplo:



```
sample_8.py X
ejemplos > clase_02 > sample_8.py > ...
1  dividendo = int(input("Ingrese el dividendo? "))
2  divisor = int(input("Ingrese el divisor? "))
3  try:
4      cociente = dividendo // divisor
5      resto = dividendo % divisor
6      print(f"El cociente de {dividendo} : {divisor} es {cociente}")
7      print(f"El resto de {dividendo} : {divisor} es {resto}")
8  except:
9      print("No se puede dividir por 0")
10

TERMINAL  PUERTOS  PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  ...  bash - clase_02  + v  [ ]  [X]  ...  ^  X
andmunoz@moitam:~/Documentos/OneDrive/Académico/INACAP/TI2041 Programacion Backend 2024-02/Ma
• terial/ti2041-2024/ejemplos/clase_02$ python sample_8.py
Ingrese el dividendo? 7
Ingrese el divisor? 0
No se puede dividir por 0
```

ERRORES Y EXCEPCIONES

- Otra forma de identificar excepciones es utilizar una sentencia **except** para **cada tipo de excepción** que puedan ocurrir.

try:

Bloque que puede provocar una excepción

except <Tipo de Exepción>:

Bloque que se ejecuta cuando ocurre ese tipo

except <Tipo de Exepción>:

Bloque que se ejecuta cuando ocurre ese tipo

except:

Bloque que se ejecuta cuando ocurre otro no identificado

ERRORES Y EXCEPCIONES

- Ejemplo:

```
sample_8.py X
ejemplos > clase_02 > sample_8.py > ...
1  dividendo = int(input("Ingrese el dividendo? "))
2  divisor = int(input("Ingrese el divisor? "))
3  try:
4      cociente = dividendo // divisor
5      resto = dividendo % divisor
6      print(f"El cociente de {dividendo} : {divisor} es {cociente}")
7      print(f"El resto de {dividendo} : {divisor} es {resto}")
8  except ZeroDivisionError:
9      print("No se puede dividir por 0")
10 except:
11     print("Ha ocurrido un error desconocido")
12
```

ERRORES Y EXCEPCIONES

- Finalmente, se puede agregar un bloque **finally** para ejecutar un bloque en **cualquier caso** de excepciones.

try:

Bloque que puede provocar una excepción

except <Tipo de Exepción>:

Bloque que se ejecuta cuando ocurre ese tipo

except:

Bloque que se ejecuta cuando ocurre otro no identificado

finally:

Bloque que se ejecuta en cualquier caso del try

ERRORES Y EXCEPCIONES

- Ejemplo:

```
sample_8.py X
ejemplos > clase_02 > sample_8.py > ...
1  dividendo = int(input("Ingrese el dividendo? "))
2  divisor = int(input("Ingrese el divisor? "))
3  try:
4      cociente = dividendo // divisor
5      resto = dividendo % divisor
6      print(f"El cociente de {dividendo} : {divisor} es {cociente}")
7      print(f"El resto de {dividendo} : {divisor} es {resto}")
8  except ZeroDivisionError:
9      print("No se puede dividir por 0")
10 except:
11     print("Ha ocurrido un error desconocido")
12 finally:
13     print("¡Programa terminado!")
14
```

FUNCIONES

- **Funciones Nativas:**

Función	Descripción	Ejemplo de Uso
<code>print(s)</code>	Imprime en la consola el valor del string s.	<code>print("hola")</code>
<code>input(s)</code>	Solicita al usuario que ingrese un valor string con un mensaje s.	<code>s = input("Valor?")</code>
<code>int(s)</code>	Convierte a entero un valor s.	<code>n = int(s)</code>
<code>float(s)</code>	Convierte a real un valor s.	<code>f = float(s)</code>
<code>str(n)</code>	Convierte a string un valor n.	<code>s = str(n)</code>

FUNCIONES

- **Más Funciones Nativas:**

Función	Descripción	Ejemplo de Uso
<code>abs(x)</code>	Devuelve el valor absoluto de x.	<code>x = abs(-7)</code>
<code>pow(x, y)</code>	Función equivalente a <code>x ** y</code> .	<code>y = x ** 2</code>
<code>max(s)</code>	Devuelve el elemento mayor de una colección.	<code>a = max([1, 5, 2])</code>
<code>min(s)</code>	Devuelve el elemento menor de una colección.	<code>b = min([1, 5, 2])</code>
<code>round(x, n)</code>	Redondea el valor x a n decimales.	<code>c = round(3.567, 1)</code>

FUNCIONES

- También se pueden traer funciones de **otras bibliotecas** que tengamos localmente en nuestro ambiente.
- Por ejemplo, una de las bibliotecas más usadas es la biblioteca **math**, la cual trae un conjunto de **funciones matemáticas** que nos pueden ayudar a resolver problemas más complejos.
- Para utilizarla, la debemos **importar** en nuestro programa:

```
from <Biblioteca> import <Función(es) a importar>
```

FUNCIONES

- Ejemplo:

```
sample_9.py ×
ejemplos > clase_02 > sample_9.py > ...
1  from math import sin, cos
2
3  # ENTRADA
4  angulo = float(input("Su ángulo (en radianes)? "))
5
6  # PROCESAMIENTO
7  sin_angulo = round(sin(angulo), 3)
8  cos_angulo = round(cos(angulo), 3)
9
10 # SALIDA
11 print("Los valores para el seno y el coseno:")
12 print(f"sin({angulo}) = {sin_angulo}")
13 print(f"cos({angulo}) = {cos_angulo}")

TERMINAL  PUERTOS  PROBLEMAS  SALIDA  ...
bash - clase_02 + v [ ] [ ] ... ^ x

andmunoz@moitam:~/Documentos/OneDrive/Académico/INACAP/TI2041 Programacion Backend 2024-02/Ma
• terial/ti2041-2024/ejemplos/clase_02$ python sample_9.py
Su ángulo (en radianes)? 3.14
Los valores para el seno y el coseno:
sin(3.14) = 0.002
cos(3.14) = -1.0
```

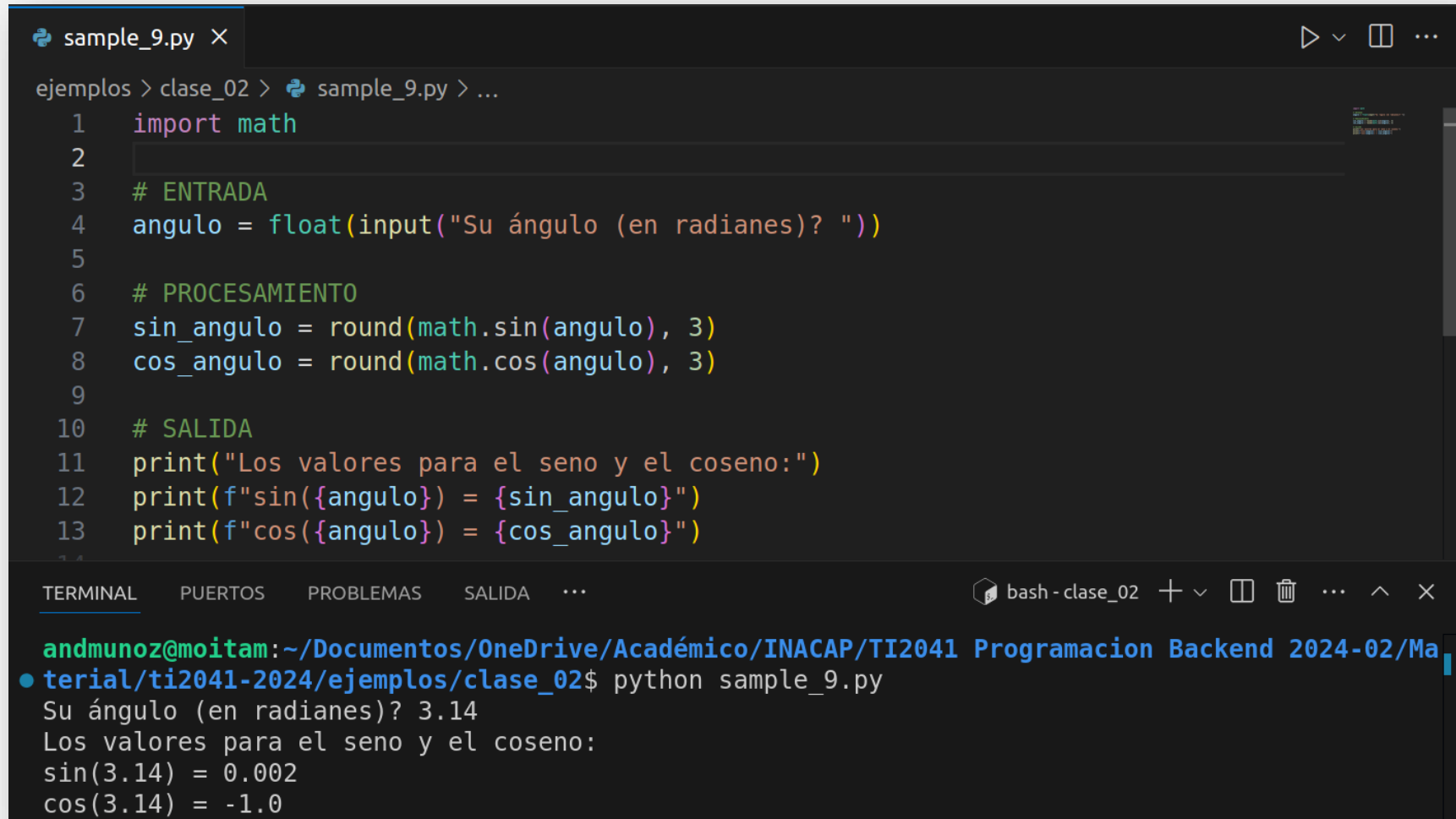
FUNCIONES

- Otra forma de hacer la importación, es utilizando **solo el nombre de la biblioteca**, sin especificar las funciones.
- En este caso, es importante considerar que **se debe distinguir de qué biblioteca estamos ejecutando la función**.
- Esta distinción se realiza **anteponiendo el nombre de la biblioteca** al nombre de la función, separándolas por un punto.

```
import <Biblioteca a importar>
```

FUNCIONES

- Ejemplo:



```
sample_9.py X
ejemplos > clase_02 > sample_9.py > ...
1 import math
2
3 # ENTRADA
4 angulo = float(input("Su ángulo (en radianes)? "))
5
6 # PROCESAMIENTO
7 sin_angulo = round(math.sin(angulo), 3)
8 cos_angulo = round(math.cos(angulo), 3)
9
10 # SALIDA
11 print("Los valores para el seno y el coseno:")
12 print(f"sin({angulo}) = {sin_angulo}")
13 print(f"cos({angulo}) = {cos_angulo}")

TERMINAL  PUERTOS  PROBLEMAS  SALIDA  ...
bash - clase_02 + v [ ] [ ] ... ^ X

andmunoz@moitam:~/Documentos/OneDrive/Académico/INACAP/TI2041 Programacion Backend 2024-02/Ma
• terial/ti2041-2024/ejemplos/clase_02$ python sample_9.py
Su ángulo (en radianes)? 3.14
Los valores para el seno y el coseno:
sin(3.14) = 0.002
cos(3.14) = -1.0
```


STRINGS

- El tipo de dato que almacena una **cadena de caracteres** en Python se conoce como **string**.
- Este tipo posee un **largo dinámico**, que permite almacenar los caracteres que uno desee.
- En general, la función **input()** devuelve la entrada por teclado siempre como **string**.
- Es posible declararlos usando **comillas simples** (' ') o **dobles** (" ") y siempre se debe ser consistentes a la hora de cerrar con el mismo tipo de comilla con el que se abrió el **string**.

STRINGS

- Algunos **operadores aritméticos** permiten realizar nuevas operaciones al tratar con **strings**.
- En particular, si existen **strings** involucrados, los operadores suma (+) y multiplicación (*) se vuelven el operador de concatenación y repetición respectivamente:
 - Sumar dos **strings** es equivalente a concatenarlos.
 - Multiplicar un **string** y un número es equivalente a repetir el string las veces que indica.
- Existen varias **funciones y métodos** que se aplican en los **strings**, y que veremos cada vez que los necesitamos.

ESTÁNDAR DE PROGRAMACIÓN

- Para mantener la legibilidad del código en Python, se establece el estándar **PEP8 – La Guía de Estilo para el Código Python**.
- En esa guía se establecen los **elementos básicos** para desarrollar nuestros programas de la mejor manera posible, considerando:
 - Code Layout
 - Uso de Strings
 - Espaciado
 - Uso de comas
 - Comentarios
 - Convenciones de Nombres
 - Recomendaciones de Programación



<https://peps.python.org/pep-0008/>

EJERCICIO EN CLASES: JALISCO

- Jalisco es un juego sencillo que aplica conocimientos de todo tipo.
- La base es que el juego le pide un número entre 1 y 100, para luego Jalisco gana siempre con el sucesor.
- Lo importante es controlar las condiciones de borde de este ejercicio:
 - Número fuera del rango
 - Ingreso de una letra en vez de un número
- Realizar el programa en Python, utilizando la consola como medio de interacción.

EN RESUMEN

- **Python** es el lenguaje que vamos a utilizar durante el curso.
- Para ello es importante recordar sus **características**:
 - De propósito general.
 - Multiparadigma.
 - Interpretado.
- Es importante utilizarlo correctamente, ya que como Back-End el comportamiento **no es el mismo**.

¿QUÉ ES DJANGO?

- **Django** es un framework de desarrollo web basado en Python.
- El objetivo principal es poder **desarrollar aplicaciones web** con éste lenguaje de programación.
- Como todo framework permite el **desarrollo rápido de aplicaciones web**, dado que simplifica el lenguaje.
- Utiliza la **arquitectura base de desarrollo MVC**.



¿QUÉ ES DJANGO?

- Fue desarrollado originalmente para **gestionar páginas web orientadas a noticias**.
- Fue liberado al público bajo una **licencia BSD** (software libre) en julio de 2005.
- El framework fue nombrado en alusión al guitarrista de jazz gitano **Django Reinhardt**.
- En junio de 2008 fue anunciado que la recién formada **Django Software Foundation** se haría cargo de Django en el futuro.



Adrian Holovaty

Simon Willison

¿QUÉ ES DJANGO?

- La meta fundamental de Django es **facilitar la creación de sitios web complejos**.
- Pone énfasis en el **re-uso, la conectividad y extensibilidad de componentes, el desarrollo rápido y el principio «DRY»** (del inglés *Don't Repeat Yourself*, «No te repitas»).
- El lenguaje **Python** es usado en todos los componentes del framework, incluso en configuraciones, archivos y en sus modelos de datos.

¿QUÉ ES DJANGO?

Ridículamente Rápido

Recargado de Funcionalidades

Seguro

Escalable

Altamente Versátil

mysite/news/models.py

```
from django.db import models
```

```
class Reporter(models.Model):  
    full_name = models.CharField(max_length=70)
```

```
    def __str__(self):  
        return self.full_name
```

```
class Article(models.Model):  
    pub_date = models.DateField()  
    headline = models.CharField(max_length=200)  
    content = models.TextField()  
    reporter = models.ForeignKey(Reporter, on_delete=models.CASCADE)
```

```
    def __str__(self):  
        return self.headline
```

¿QUÉ ES DJANGO?

- Algunas empresas que han utilizado Django:



Bitbucket



Platzi



Instagram
(al principio)



Mozilla



Pinterest
(al principio)



Disquis



Udemy



Spotify

¿QUÉ ES DJANGO?

- El **sitio oficial** (en inglés):

<https://www.djangoproject.com/>

- Además posee **documentación** en diferentes idiomas.
- Aquí está la documentación **en español** (parcialmente):

<https://docs.djangoproject.com/es/5.1/>

CÓMO INSTALAR DJANGO

- **Pre-requisitos:**

- Python 3.10 o superior (para Django 5).

<https://www.python.org/downloads/>

- Package Installer for Python (PIP).

<https://pypi.org/project/pip/>

CÓMO INSTALAR DJANGO

- **Instalación de Django:**

- Abra una ventana de la línea de comandos (Windows) o una terminal (Linux y MacOS).
- Escriba el comando de instalación:

```
$ python -m pip install Django
```

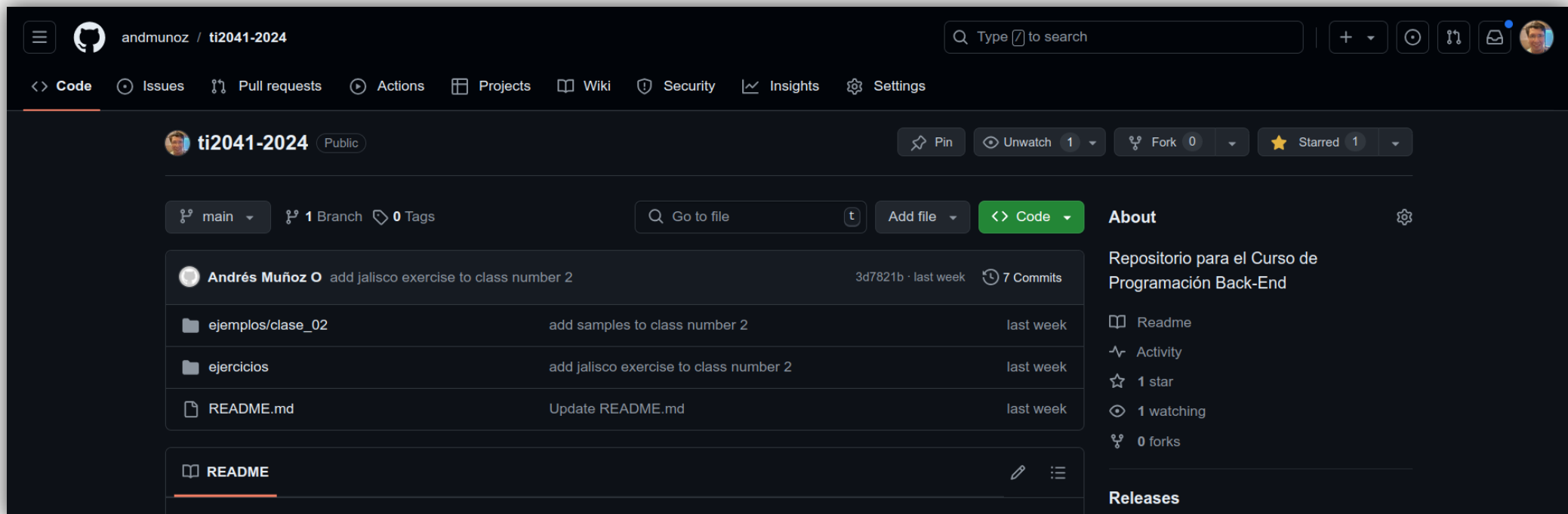
- También se puede usar el alias de Python **py** en vez de la instrucción **python** o simplemente la instrucción **pip** si el sistema lo permite.

<https://docs.djangoproject.com/es/5.1/intro/install/>

CÓMO CREAR UN PROYECTO

- **Pre-requisitos:**

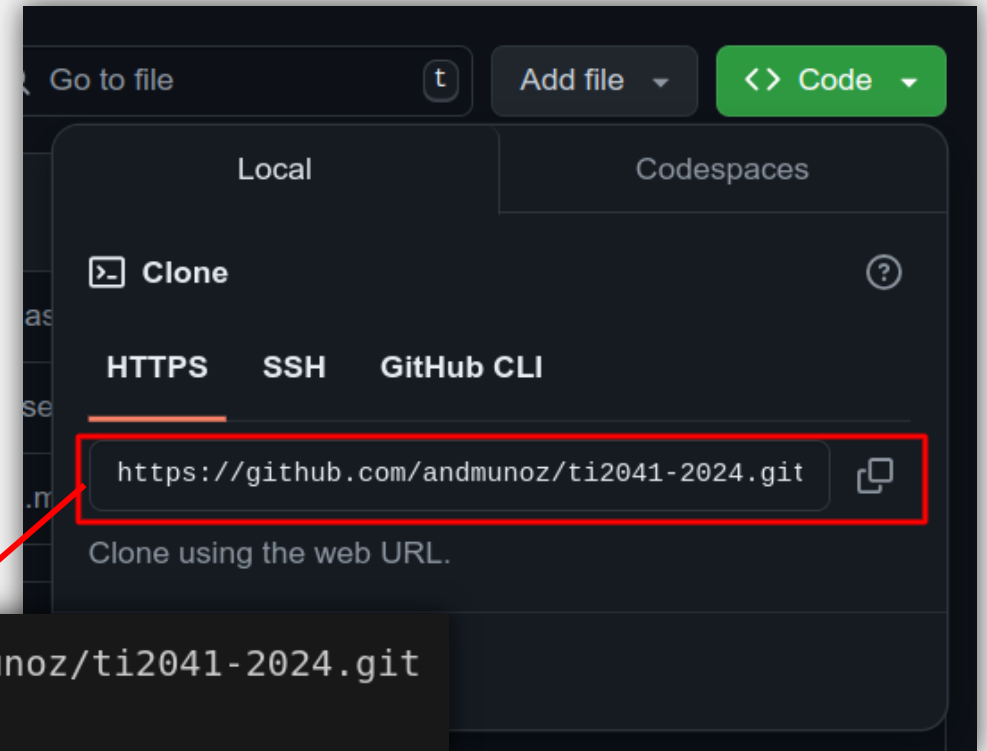
- Debemos recordar que trabajaremos en un repositorio GitHub.
- Por lo tanto, debemos crear nuestro repositorio **ti2041-2024**.



CÓMO CREAR UN PROYECTO

- **Pre-requisitos:**

- Luego, localmente, debemos clonar el repositorio.
- No importa que esté vacío.

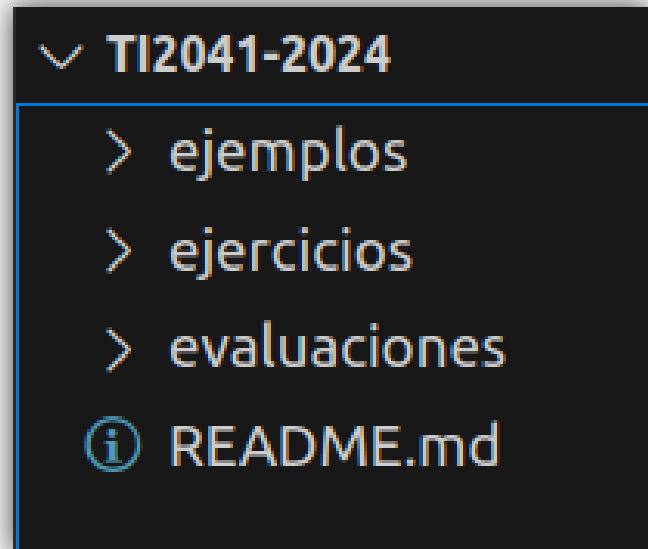


```
• andmunoz@moitam:~/temp$ git clone https://github.com/andmunoz/ti2041-2024.git
Clonando en 'ti2041-2024'...
remote: Enumerating objects: 31, done.
remote: Counting objects: 100% (31/31), done.
remote: Compressing objects: 100% (24/24), done.
remote: Total 31 (delta 3), reused 23 (delta 1), pack-reused 0 (from 0)
Recibiendo objetos: 100% (31/31), 5.97 KiB | 2.99 MiB/s, listo.
Resolviendo deltas: 100% (3/3), listo.
```


CÓMO CREAR UN PROYECTO

- **Pre-requisitos:**

- Finalmente, creamos nuestras carpetas en el repositorio local:
 - ejemplos
 - ejercicios
 - Evaluaciones
- Recuerde que si al crear el repositorio, solicitó crear el archivo **Readme.md**, aparecerá también localmente.



CÓMO CREAR UN PROYECTO

- **Creando el Proyecto:**

- Para la creación del proyecto, debemos entrar primero a la carpeta raíz donde haremos el proceso.
- En este caso, usemos la carpeta **ejemplos** de nuestro repositorio local.
- Así que vamos a dicha carpeta en un terminal y ejecutamos el comando para crear un proyecto:

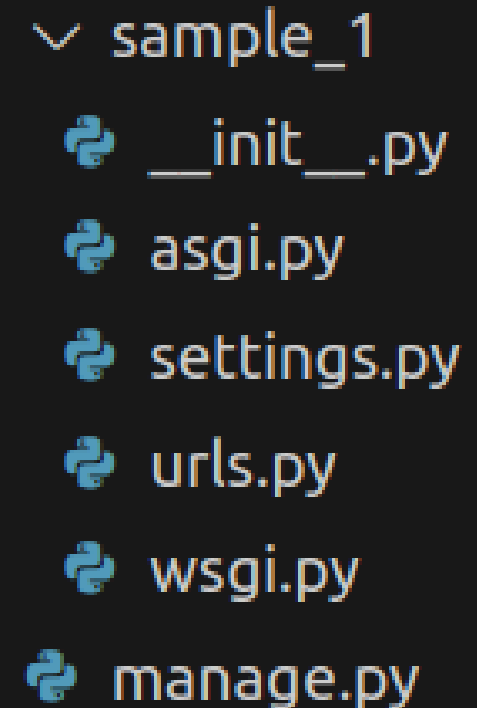
```
$ django-admin startproject sample_1
```

- Nota: Si no funciona este comando, puede usar **python -m django startproject sample_1**

<https://docs.djangoproject.com/es/5.1/intro/tutorial01/>

ESTRUCTURA DE UN PROYECTO

- La carpeta del proyecto `sample_1` contiene dos elementos:
 - **Una carpeta `sample_1`** (mismo nombre).
 - Corresponde a la carpeta que contendrá su aplicación web.
 - **Un archivo `manage.py`**.
 - Corresponde al programa utilitario que le permite interactuar con su proyecto Django.

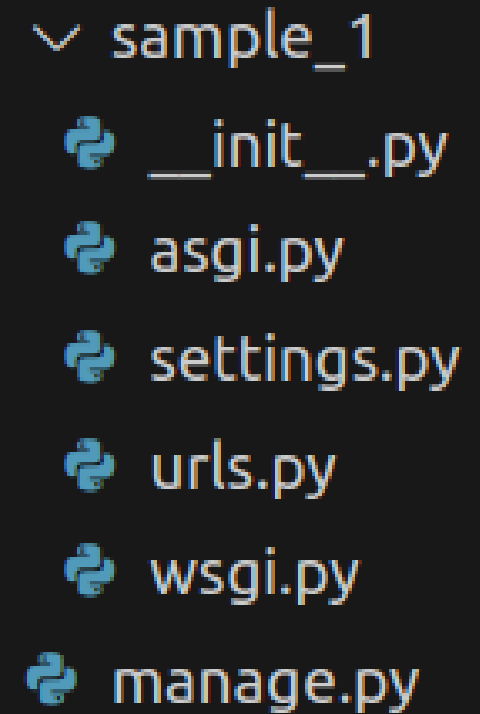


```
▼ sample_1
  __init__.py
  asgi.py
  settings.py
  urls.py
  wsgi.py
  manage.py
```

ESTRUCTURA DE UN PROYECTO

- Dentro de la carpeta de la aplicación, se encuentran 5 archivos:

- **`__init__.py`**: Un archivo vacío que le indica a Python que este directorio debería ser considerado como un paquete Python.
- **`setting.py`**: Ajustes/configuración para este proyecto Django.
- **`urls.py`**: Las declaraciones URL para este proyecto Django, es decir una tabla de contenidos de su sitio basado en Django.
- **`asgi.py`** y **`wsgi.py`**: Un punto de entrada para servidores web compatibles con WSGI y ASGI.



```
▼ sample_1
  🐍 __init__.py
  🐍 asgi.py
  🐍 settings.py
  🐍 urls.py
  🐍 wsgi.py
  🐍 manage.py
```

ESTRUCTURA DE UN PROYECTO

- **Ejecutando el Proyecto:**

- Cuando uno quiere "ejecutar" el proyecto, en realidad lo que hace es iniciar un servidor de desarrollo con la aplicación.
- Para ello, se debe ir al directorio del proyecto (al raíz) y ejecutar la siguiente instrucción:

```
o s/clase_03/sample_1$ python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

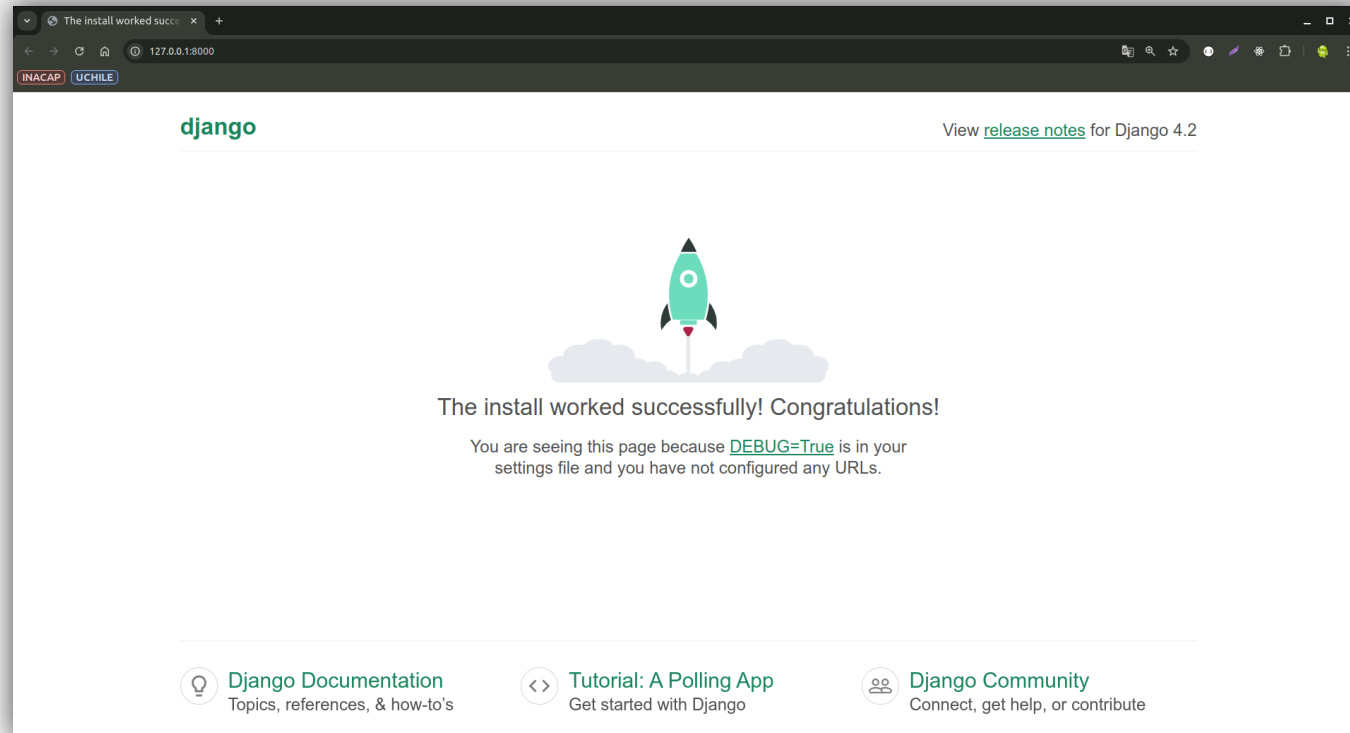
System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin
, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
August 26, 2024 - 13:33:31
Django version 4.2.5, using settings 'sample_1.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

ESTRUCTURA DE UN PROYECTO

- **Ejecutando el Proyecto:**

- Luego hay que abrir un navegador con la url: <http://127.0.0.1:8000/>



EJEMPLO PRÁCTICO

- Hagamos alguna modificación a nuestro proyecto que nos permita descubrir algunas particularidades:
 - Cree un proyecto Django en la carpeta ejemplos llamados **sample_2**.
 - Ejecute el siguiente comando antes de probar su aplicación:

```
sample_2$ python manage.py startapp exampleapp
```

- Ejecute el servidor de desarrollo (**runserver**) de la aplicación.
- Abra el navegador.

¿Qué ve ahora?

EJEMPLO PRÁCTICO

- Ahora realice los siguientes cambios (sin bajar el servidor):
 - Abra el archivo `exampleapp/views.py` y haga el siguiente cambio:

```
1  from django.shortcuts import render, HttpResponse
2
3  # Create your views here.
4  def index(request):
5      |   return HttpResponse("Hola a todos. Esta es mi primera aplicación en Django")
```

- **Nota:** Recuerde que estamos construyendo lo que va a ver el usuario en el navegador.

EJEMPLO PRÁCTICO

- Ahora... (cont):
 - Luego, agregar el archivo `exampleapp/urls.py` con lo siguiente:

```
1  from django.urls import path
2
3  from . import views
4
5  urlpatterns = [
6      path("", views.index, name="index"),
7  ]
```

- **Nota:** este archivo solo tiene las rutas de la aplicación y no del proyecto.

EJEMPLO PRÁCTICO

- Ahora... (cont):
 - Finalmente, modifique `sample_2/urls.py` con lo siguiente:

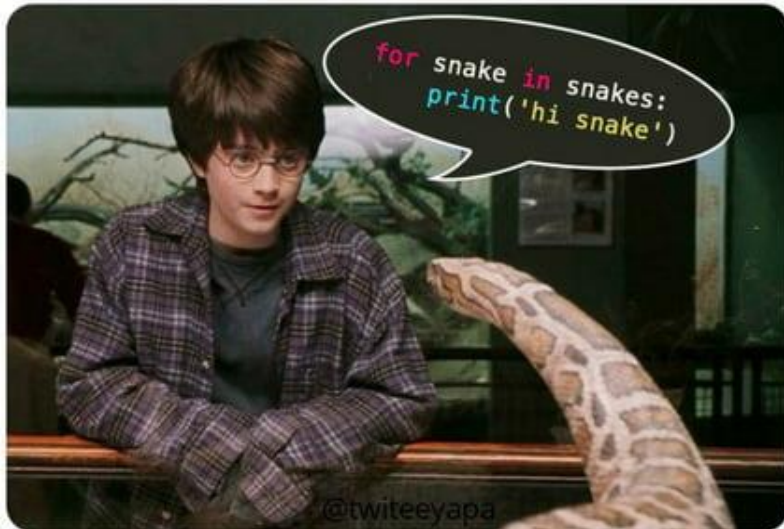
```
17 from django.contrib import admin
18 from django.urls import path, include
19
20 urlpatterns = [
21     path('exampleapp/', include('exampleapp.urls')),
22     path('admin/', admin.site.urls),
23 ]
```

- **Nota:** Este archivo contiene las rutas del proyecto

EN RESUMEN

- Django es un **framework de desarrollo de aplicaciones web**, rápido, seguro y con un montón de funciones incorporadas que la hacen fácil de desarrollar.
- El objetivo principal es que los desarrolladores puedan **confeccionar aplicaciones con poco esfuerzo y en tiempos reducidos**, satisfaciendo la demanda de sus clientes.
- Como utiliza Python, se puede utilizar en **múltiples escenarios**, los cuales dependen de las necesidades propias de cada negocio.

That's how Harry could talk to snakes



UNIVERSIDAD TECNOLÓGICA DE CHILE
INSTITUTO PROFESIONAL
CENTRO DE FORMACIÓN TÉCNICA