



PYTHON

Manejo de excepciones y control de errores

1

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Detección y resolución de errores de sintaxis
 - Detectados por el intérprete.
 - Fáciles de detectar.

```
lista.remove(candidato  
print(lista)
```

Error (falta el paréntesis de cierre)

```
File "d\022-adivinador-planetas.py", line 15
```

Línea del detección
del error

```
    print(lista)
```

```
    ^
```

^ Indica la posición posterior a la
detección del error

```
SyntaxError: invalid syntax
```

Tipo de error

Descripción del error

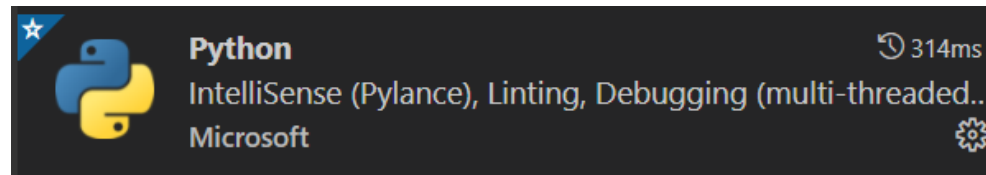
MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Detección y resolución de **errores lógicos**
 - No generan un mensaje de error.
 - Provocan que el programa no funcione de acuerdo a los requisitos.
 - Detectados durante las pruebas: difíciles de detectar.

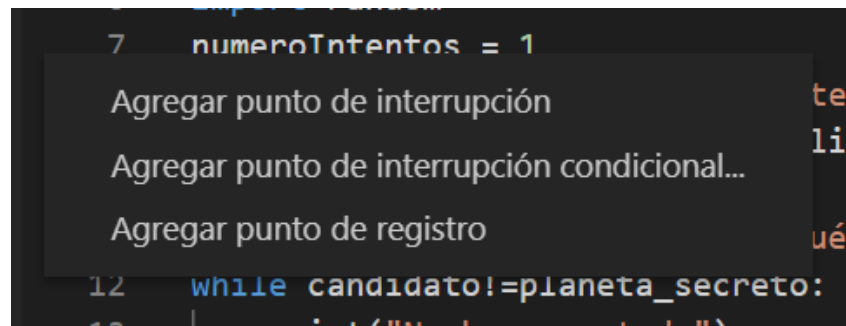
```
>>> if edad<=18:  
...     print("Es mayor de edad")
```

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Detección y resolución de errores lógicos
 - Técnica detección: ejecución paso a paso.

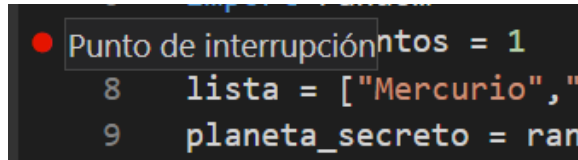


- Agregar puntos de interrupción (breakpoints):
 - Botón derecho a la izquierda de la línea.



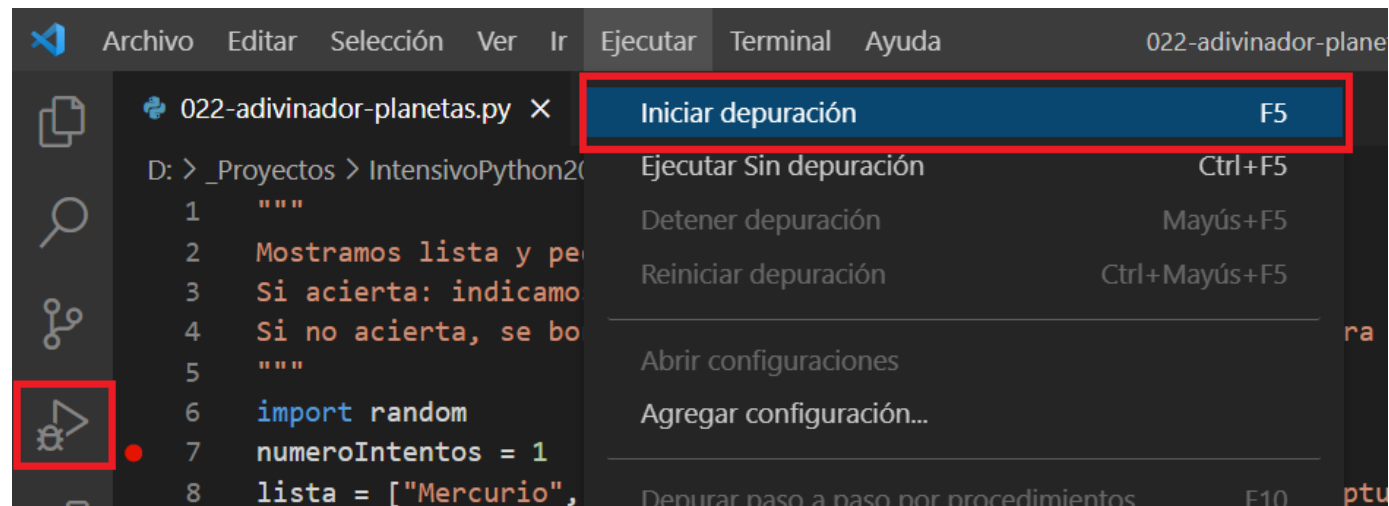
MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Detección y resolución de errores lógicos
 - Se pueden activar y desactivar pulsando sobre el indicador rojo.
 - Ejecutar en modo de depuración.



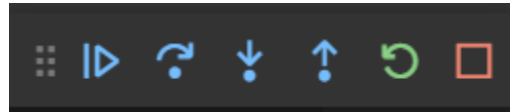
Punto de interrupción

```
8 lista = ["Mercurio",  
9 planeta_secreto = ran
```



MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Detección y resolución de errores lógicos



- Continuar (hasta el siguiente breakpoint) (F5)
- Depurar paso a paso por procedimientos (F10)
- Depurar paso a paso por instrucciones (F11)
- Salir de la depuración (Mayúsculas+F11)
- Reiniciar (Control+Mayúsculas+F5)
- Detener (Mayúsculas+F5)

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Detección y resolución de errores lógicos:
 - Inspección de variables

The screenshot shows a Python IDE interface. On the left, a sidebar titled 'EJECUCIÓN Y DEPURACIÓN' (Execution and Debugging) contains a 'VARIABLES' section. Under 'Locals', 'numero_dias_vacaciones' is set to 5. Under 'Globals', 'DIAS_TOTALES' is 20, 'laborales' is a tuple of days, and 'vacaciones_solicitadas' is 5. The main editor shows a Python script with the following code:

```
C: > GoogleDrive > _Transparencias_propias > Programación en Python Oficial de Python
1  laborales = ("Lunes", "Martes", "Miércoles", "Jueves", "Viernes")
2  DIAS_TOTALES = 20
3  def calcular_vacaciones(numero_dias_vacaciones):
4      dias_pendientes = DIAS_TOTALES - numero_dias_vacaciones
5      return dias_pendientes
6
7  vacaciones_solicitadas = 5
8  pendientes = calcular_vacaciones(vacaciones_solicitadas)
9  print(pendientes)
```

A yellow cursor is positioned at the start of line 4, and a red dot is on line 7.

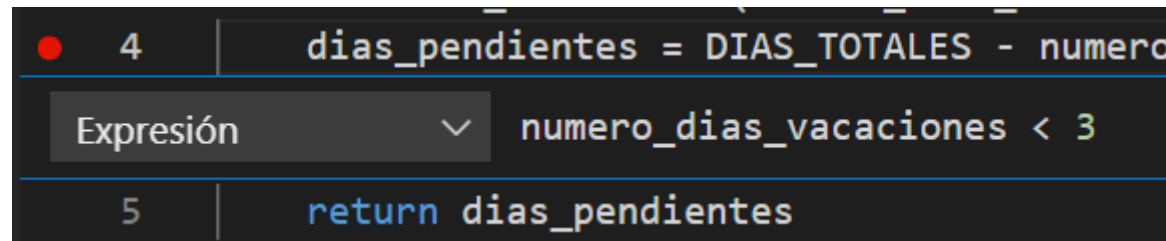
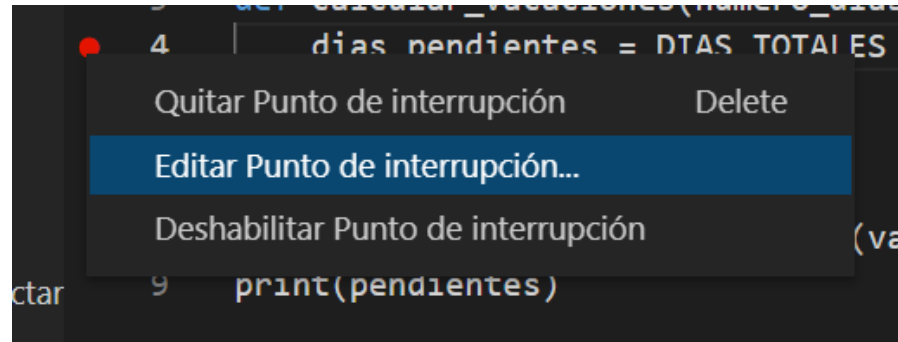
MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Detección y resolución de errores lógicos:
 - Inspección de variables

```
leDrive > _transparencias_propias > Programacion en Py  
laborales = ("Lunes", "Martes", "Miércoles", "  
DIAS_TOTALES = 20  
def calcular_vacaciones(numero_dias_vacacio  
    dias_pendientes = DIAS_TOTALES - numero  
    return dias_pendientes  
vacaciones_solicitadas = 5
```


MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Detección y resolución de errores lógicos:
 - Breakpoints condicionados:



MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Detección y resolución de **errores de ejecución**
 - Se producen en ejecución, como consecuencia del contexto.

```
a=5
```

```
b=0
```

```
resultado = a/b
```

Ubicación del error

```
File "programa.py", line 3, in <module>
```

```
resultado = a/b
```

```
ZeroDivisionError: division by zero
```

Tipo de error

Descripción del error

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Control de errores: try-except
 - Concepto de exception
 - Permiten controlar errores previstos:

```
try:  
    bloque de código  
except:  
    bloque de código
```

- Tipos de excepciones:
 - <https://docs.python.org/3/library/exceptions.html>

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- **Control de errores: try-except**

- **Especificiando el error:**

```
try:  
    bloque de código  
except ZeroDivisionError:  
    bloque de código
```

- **Especificiando el error, múltiples niveles:**

```
try:  
    bloque de código  
except ZeroDivisionError:  
    bloque de código  
except:  
    bloque de código
```

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- **Control de errores: try-except**
 - **Especificando el error, múltiples excepciones, múltiples niveles:**

```
try:  
    bloque de código  
except (ZeroDivisionError, ValueError):  
    bloque de código  
except:  
    bloque de código
```

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- **Control de errores: try-except**
 - **Bloque else: se ejecuta si no se produce ningún error.**

```
try:  
    bloque de código  
except:  
    bloque de código  
else:  
    bloque de código
```

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Control de errores: try-except
 - Bloque finally: se ejecuta siempre.

```
try:  
    bloque de código  
except:  
    bloque de código  
finally:  
    bloque de código
```

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Control de errores: try-except
 - Combinación

```
try:  
    bloque de código  
except:  
    bloque de código  
else:  
    bloque de código  
finally:  
    bloque de código
```


MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Manejo de excepciones
 - Generación:
 - Función `raise()`
 - Permite generar y propagar una excepción.
`raise ValueError()`
`raise ValueError` → Forma abreviada
 - Permite propagar una excepción.
`try:`
`...`
`except ValueError:`
`raise`

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Manejo de excepciones

- Generación:

- Función raise()

- Permite apilar y propagar una excepción.

```
def calcular_edad(anyo_nacimiento, anyo_actual):  
    if type(anyo_nacimiento) is not int:  
        raise TypeError  
  
try:  
    funcion()  
except TypeError as exception:  
    raise RuntimeError('Error en la funcion') from exception
```

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Manejo de excepciones

- Excepciones propias (sin información):

```
class LongitudInsuficienteError(Exception):  
    pass  
  
def funcion():  
    raise LongitudInsuficienteError()  
  
try:  
    funcion()  
except LongitudInsuficienteError as err:  
    print("Ha ocurrido un error")
```

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Manejo de excepciones

- Excepciones propias (con información):

```
class LongitudInsuficienteError(Exception):  
    def __init__(self, mensaje):  
        self.mensaje = mensaje  
        super().__init__(self.mensaje)  
  
def funcion():  
    raise LongitudInsuficienteError("Longitud insuficiente")  
  
try:  
    funcion()  
except LongitudInsuficienteError as err:  
    print("Ha ocurrido un error:" + str(err))
```

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Diseño de código robusto:
 - Test
 - Documentación
 - Sencillez
 - Limpieza
 - Convenciones
 - Uso de constantes
 - Métodos y funciones
 - Orientación a Objetos

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

Test unitarios

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Test unitarios
 - Frameworks de pruebas incluidos en Python:
 - unittest
 - <https://docs.python.org/3.9/library/unittest.html>
 - <https://docs.python.org/3.9/library/unittest.html#classes-and-functions>
 - doctest
 - <https://docs.python.org/3/library/doctest.html>

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

unittest

<https://docs.python.org/3/library/unittest.html>

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Módulo unittest
- La clase que implementa los test debe heredar de unittest.TestCase
- Métodos: los métodos de test comienzan por test_

```
@classmethod
def setUpClass(cls) -> None:
    print("Se ejecuta antes de iniciar el proceso de test")

@classmethod
def tearDownClass(cls) -> None:
    print("Se ejecuta después de iniciar el proceso de test")

def setUp(self) -> None:
    print("Se ejecuta antes de ejecutar cada test")

def tearDown(self) -> None:
    print("Se ejecuta después de ejecutar cada test")

def test_prueba(self):
    self.assertEqual(3,4)
```

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Test unitarios
 - Métodos de validación más usados.

| Method | Checks that | New in |
|--|-----------------------------------|--------|
| <code>assertEqual(a, b)</code> | <code>a == b</code> | |
| <code>assertNotEqual(a, b)</code> | <code>a != b</code> | |
| <code>assertTrue(x)</code> | <code>bool(x) is True</code> | |
| <code>assertFalse(x)</code> | <code>bool(x) is False</code> | |
| <code>assertIs(a, b)</code> | <code>a is b</code> | 3.1 |
| <code>assertIsNot(a, b)</code> | <code>a is not b</code> | 3.1 |
| <code>assertIsNone(x)</code> | <code>x is None</code> | 3.1 |
| <code>assertIsNotNone(x)</code> | <code>x is not None</code> | 3.1 |
| <code>assertIn(a, b)</code> | <code>a in b</code> | 3.1 |
| <code>assertNotIn(a, b)</code> | <code>a not in b</code> | 3.1 |
| <code>assertIsInstance(a, b)</code> | <code>isinstance(a, b)</code> | 3.2 |
| <code>assertNotIsInstance(a, b)</code> | <code>not isinstance(a, b)</code> | 3.2 |

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Test unitarios

- Otros métodos:

- unittest.main() → Ejecuta los test de la clase
 - assertRaise → Comprueba que lanza una excepción.
 - fail → Provoca un error forzado.
 - skipTest → Hace que se salte el test.
 - self.skipTest("No aplica")

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Test unitarios
 - unittest. Ejemplo: código a probar. calculadora.py

```
def sumar(s1, s2):  
    return s1+s2
```

```
def restar(r1, r2):  
    return r1*r2
```

```
def dividir(d1, d2):  
    return d1/d2
```

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Test unitarios

- Unittest. Ejemplo: test unitario.

```
import unittest
import calculadora

class TestCalculadora(unittest.TestCase):
    def test_suma(self):
        self.assertEqual(calculadora.sumar(10.3, 4.1), 14.4)

    def test_restar(self):
        self.assertEqual(calculadora.restar(8,2), 6)

    def test_dividir(self):
        self.assertEqual(calculadora.dividir(8,0), 4)

if __name__ == "__main__":
    unittest.main()
```

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

doctest

<https://docs.python.org/3/library/doctest.html>

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Test unitarios
 - Doctest: incrustados en los comentarios

```
def sumar(s1, s2):  
    """  
    Suma dos números y devuelve el resultado  
    >>> sumar(3,4)  
    15  
    """  
    return s1+s2
```

MANEJO DE EXCEPCIONES Y CONTROL DE ERRORES

- Test unitarios
 - Doctest: ejecución desde la ejecución.
 - **python -m doctest -v calculadora.py**
 - Doctest: ejecución desde el código. Sin el parámetro `verbose=True` sólo muestra los errores por la consola.

```
import doctest
doctest.testmod(verbose=True)
```

```
> python calculadora.py
```