Types of Errors

In Python, there are several types of errors that can occur during the execution of a program. These errors are categorized into three main types:

1. **Syntax Errors**:

- Syntax errors, also known as parsing errors, occur when the code violates the syntax rules of Python.

- These errors prevent the code from being executed.

Example:

```
print("Hello, World!)
```

Output:

```
File "<stdin>", line 1
  print("Hello, World!)
                      ^
SyntaxError: EOL while scanning string literal
```

Explanation:

- In this example, a closing quotation mark (`"`) is missing in the `print` statement. This violates the syntax rules, causing a `SyntaxError`.

2. **Logical Errors**:

- Logical errors occur when the code is syntactically correct, but it does not produce the expected output due to a mistake in the logic of the program.

Example:

```
def add_numbers(a, b):
    return a * b

result = add_numbers(2, 3)
print(result)
```

Output:

```
6
```

Explanation:

- In this example, the `add_numbers` function is supposed to add two numbers, but it is actually multiplying them. This is a logical error.

3. **Runtime Errors (Exceptions):**

- Runtime errors, also known as exceptions, occur during the execution of a program. They are not detected by the Python interpreter until the code is actually running.

- There are various types of runtime errors, such as `ZeroDivisionError`, `NameError`, `TypeError`, `ValueError`, etc.

Example:

```
num1 = 10
num2 = 0

result = num1 / num2
print(result)
```

Output:

```
Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
ZeroDivisionError: division by zero
```

Explanation:

- In this example, we are trying to divide `num1` by `num2`, where `num2` is `0`. This causes a `ZeroDivisionError` because dividing by zero is undefined in mathematics.

4. **Semantic Errors:**

- Semantic errors are a bit more subtle and harder to detect. They occur when the code is grammatically correct, and it runs without giving any error messages, but it does not produce the desired output.

Example:

```
def calculate_average(numbers):
```

```
    total = 0
    for number in numbers:
        total = total + number
    average = total / len(numbers)
    return average

numbers = [10, 20, 30, "40"]
result = calculate_average(numbers)
print(result)
```

Output:

```
Traceback (most recent call last):
  File "<stdin>", line 9, in <module>
  File "<stdin>", line 5, in calculate_average
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Explanation:

- In this example, the `calculate_average` function expects a list of numbers, but one of the elements in the list is a string (`"40"`). This causes a `TypeError` when trying to add an integer to a string.

Understanding these different types of errors can help you debug and improve your Python programs.