

# Errors

# Types of errors

These are common to all programming languages:

- Logic errors
- Syntax errors
- Runtime errors

# Logic errors

# Logic errors

A program has a logic error if it does not behave as expected. Typically discovered via failing tests or bug reports from users.

Spot the logic error:

```
# Sum up the numbers from 1 to 10
sum = 0
x = 1
while x < 10:
    sum += x
    x += 1
```

To avoid the wrath of angry users, write tests.

# Syntax errors

# Syntax errors

Each programming language has syntactic rules. If the rules aren't followed, the program cannot be parsed and will not be executed at all.

Spot the syntax errors:

```
if x > 5
    x += 1
```

```
sum = 0
x = 0
while x < 10:
    sum + = x
    x + = 1
```

To fix a syntax error, read the message carefully and go through your code with a critical eye.

# Syntax errors

Each programming language has syntactic rules. If the rules aren't followed, the program cannot be parsed and will not be executed at all.

Spot the syntax errors:

```
if x > 5 # Missing colon
    x += 1
```

```
sum = 0
x = 0
while x < 10:
    sum + = x
    x + = 1
```

To fix a syntax error, read the message carefully and go through your code with a critical eye.

# Syntax errors

Each programming language has syntactic rules. If the rules aren't followed, the program cannot be parsed and will not be executed at all.

Spot the syntax errors:

```
if x > 5 # Missing colon
    x += 1
```

```
sum = 0
x = 0
while x < 10:
    sum += x # No space needed between + and =
    x += 1
```

To fix a syntax error, read the message carefully and go through your code with a critical eye.



# SyntaxError

What it technically means:

The file you ran isn't valid python syntax

What it practically means:

You made a typo

What you should look for:

- Extra or missing parenthesis
- Missing colon at the end of an if, while, def statements, etc.
- You started writing a statement but forgot to put any clauses inside

Examples:

```
print("just testing here"))
```

```
title = 'Hello, ' + name ', how are you?'
```

# IndentationError/TabError

What it technically means:

The file you ran isn't valid Python syntax, due to indentation inconsistency.

What it sometimes means:

You used the wrong text editor (or one with different settings)

What you should look for:

- A typo or misaligned block of statements
- A mix of tabs and spaces
  - Open your file in an editor that shows them
  - `cat -A filename.py` will show them

Example:

```
def sum(a, b):  
    total = a + b  
    return total
```

# Runtime errors

# Runtime errors

A runtime error happens while a program is running, often halting the execution of the program. Each programming language defines its own runtime errors.

Spot the runtime error:

```
def div_numbers(dividend, divisor):  
    return dividend/divisor  
  
quot1 = div_numbers(10, 2)  
quot2 = div_numbers(10, 1)  
quot3 = div_numbers(10, 0)  
quot4 = div_numbers(10, -1)
```

To prevent runtime errors, code defensively and write tests for all edge cases.

# Runtime errors

A runtime error happens while a program is running, often halting the execution of the program. Each programming language defines its own runtime errors.

Spot the runtime error:

```
def div_numbers(dividend, divisor):  
    return dividend/divisor  
  
quot1 = div_numbers(10, 2)  
quot2 = div_numbers(10, 1)  
quot3 = div_numbers(10, 0) # Cannot divide by 0!  
quot4 = div_numbers(10, -1)
```

To prevent runtime errors, code defensively and write tests for all edge cases.

# TypeError: 'X' object is not callable

What it technically means:

Objects of type X cannot be treated as functions

What it practically means:

You accidentally called a non-function as if it were a function

What you should look for:

- Parentheses after variables that aren't functions

Example:

```
sum = 2 + 2  
sum(3, 5)
```

# ...NoneType...

What it technically means:

You used None in some operation it wasn't meant for

What it practically means:

You forgot a return statement in a function

What you should look for:

- Functions missing return statements
- Printing instead of returning a value

Example:

```
def sum(a, b):  
    print(a + b)  
  
total = sum( sum(30, 45), sum(10, 15) )
```

# NameError

What it technically means:

Python looked up a name but couldn't find it

What it practically means:

- You made a typo
- You are trying to access variables from the wrong frame

What you should look for:

- A typo in the name
- The variable being defined in a different frame than expected

Example:

```
fav_nut = 'pistachio'  
best_chip = 'chocolate'  
trail_mix = Fav_Nut + best__chip
```



# UnboundLocalError

What it technically means:

A variable that's local to a frame was used before it was assigned

What it practically means:

You are trying to both use a variable from a parent frame, and have the same variable be a local variable in the current frame

What you should look for:

Assignments statements after the variable name

Example:

```
sum = 0

def sum_nums(x, y):
    sum += x + y
    return sum

sum_nums(4, 5)
```

# TraceBacks

# What's a traceback?

When there's a runtime error in your code, you'll see a **traceback** in the console.

```
def div_numbers(dividend, divisor):  
    return dividend/divisor  
  
quot1 = div_numbers(10, 2)  
quot2 = div_numbers(10, 1)  
quot3 = div_numbers(10, 0)  
quot4 = div_numbers(10, -1)
```

```
Traceback (most recent call last):  
  File "main.py", line 14, in <module>  
    quot3 = div_numbers(10, 0)  
  File "main.py", line 10, in div_numbers  
    return dividend/divisor  
ZeroDivisionError: division by zero
```

# Parts of a Traceback

- The error message itself
- Lines #s on the way to the error
- What's on those lines

The most recent line of code is always last (right before the error message).

```
Traceback (most recent call last):  
  File "main.py", line 14, in <module>  
    quot3 = div_numbers(10, 0)  
  File "main.py", line 10, in div_numbers  
    return dividend/divisor  
ZeroDivisionError: division by zero
```

# Reading a Traceback

1. Read the error message (remember what common error messages mean!)
2. Look at **each line**, bottom to top, and see if you can find the error.

```
Traceback (most recent call last):  
  File "main.py", line 14, in <module>  
    quot3 = div_numbers(10, 0)  
  File "main.py", line 10, in div_numbers  
    return dividend/divisor  
ZeroDivisionError: division by zero
```

# Fix this code!

```
def f(x):  
    return g(x - 1)  
  
def g(y):  
    return abs(h(y) - h(1 / y))  
  
def h(z):  
    z * z  
  
print(f(12))
```