Introduction to Python

T5 Bootcamp by SDAIA

SDAIA
الهيئة السعودية للبيانات والذكاء الاصطناعي
Saudi Data & AI Authority

Exceptions

SDAIA
الهيئة السعودية للبيانات والذكاء الاصطناعي
Saudi Data & AI Authority

Outline

- Categories of Errors
  - Syntax Errors
  - Logical Errors
  - Runtime Errors
- Exceptions
  - Built-in Exceptions
  - User-defined Exceptions
  - Single except
  - Multiple except
- Errors are your friends

Three Categories of Errors

1. Syntax Errors
2. Logical Errors
3. Runtime Errors

Syntax Errors

- ***Syntax errors*** occur when the code you've written does not follow the correct syntax or **grammar** of the programming language.
- Syntax errors are caused by missing parentheses, quotes, or semicolons, or by using an incorrect keyword or operator.

```
# Missing colon after `if`
if (x > 5)
    print("x")

# Unbalanced quotes
print("x

# Invalid variable name
my-variable = 10
```

Causes of Indentation Errors in Python

- **Mixing tabs and spaces**: Python is strict about using either spaces or tabs, but not both, for indentation.
- Accidentally using both can lead to errors, especially when **copying and pasting code** from different sources.
- It an easily happen while **refactoring** (reorganizing code).

```python
# No indentation
if (x > 5):
print("x")

# Uneven indentation: 5 spaces
if (x > 5):
        print("x") # 5

# Uneven indentation: 3 spaces
if (x > 5):
   print("x")

# Uneven indentation: 1 space
if (x > 5):
 print("x")
```

Logical Errors

- ***Logical errors*** occur when the code runs **without raising an exception** but does not produce the expected output.
- These errors are often caused by errors in the algorithm or the logic of the program.
- Logical errors are difficult to detect and can be caused by incorrect assumptions about the data, incorrect calculations, or incorrect control structures.

Examples of Logical Errors

```
# Using the wrong function for the job
square = math.sqrt(x)

# Checking for equality with a string literal
if (x == "10"):
    print("x is equal to 10")

# function implementation error
def is_even(number):
  if number % 2 != 0: # <-- inverted condition
    return True
  else:
    return False
```

```
# Unintended indentation levels
for i in range(10):
    for j in range(10):
        print(i)
    print(j) # <-- indent inside

# Infinite loop
condition = 5 < 10
while condition:
    print("always true")
```

Runtime Errors

- ***Runtime errors*** occur when the code is executed, and an unexpected event occurs that interrupts the normal flow of the program.
- These errors are often caused by external factors such as incorrect user input, unavailable resources, or network errors.

Examples of Runtime Errors

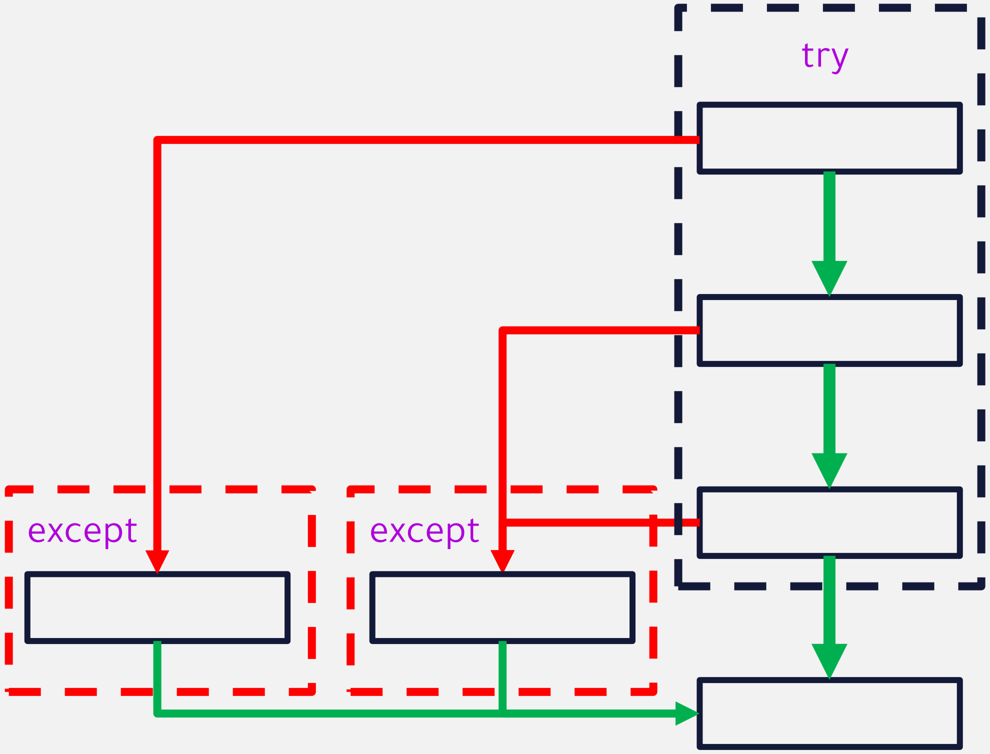
```python
# Trying to divide by zero
y = 0
x = 10 / y

# Parsing errors: user enters a word instead
x = int(input('Please enter a number: '))

# Operations on incompatible types
x = 10 + '5'

# Access an attribute of a non-object (None)
y = my_variable.names

# Passing wrong values or types
a = 4; b = '3'
add_numbers(a, b)
```

```python
# Reading a file that does not exist
file = open('nofile.txt', 'r')

# Access a non-existent index
my_list = [1, 2, 3]
my_list[3]

# Access a non-existent key
my_dict = {'a': 1, 'b': 2}
my_dict['c']

# Access a non-existent attribute
my_object = Employee('John')
my_object.weight
```

Exceptions

Errors in Python whether being bugs in coding or valid error cases are called Exceptions.

Exceptions interrupt the ideal happy path of the program, usually, because it is interacting with the real-world; be it: user input, system files, network, or other programs.



Types of Exceptions

- In Python, exceptions can be divided into two main categories:
  - Built-in exceptions
  - User-defined exceptions

Built-in Exceptions

SyntaxError: Raised when there is a syntax error in the code.

IndentationError: Raised when there is an incorrect indentation in the code.

NameError: Raised when a variable or function is used before it has been defined.

TypeError: Raised when a function or operation is applied to an object of the wrong type.

ValueError: Raised when a function or operation is applied to an object of the correct type but with an invalid value.

ZeroDivisionError: Raised when division by zero occurs.

User-defined Exceptions

- User-defined exceptions are useful when you want to raise an exception that is specific to your application or domain.

```python
class MyCustomError(Exception):
    pass
```

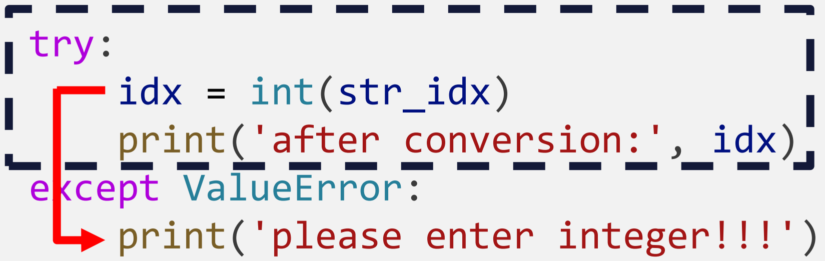
Handling Exceptions

Coding Error (Bug)

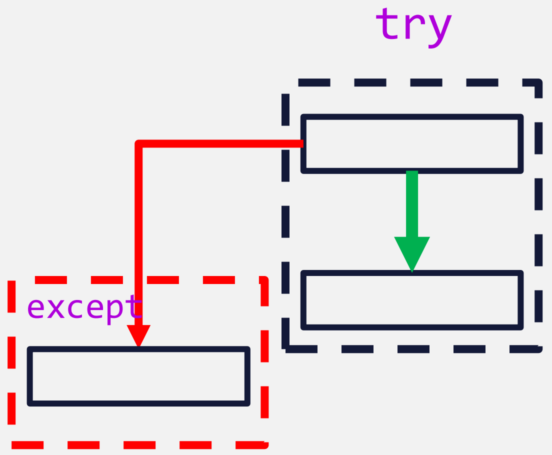
Valid Error (Solvable by Code)

Exception Type	Cause	How to handle
SyntaxError **IndentationError** **NameError**	Incorrect code syntax	Fix Code
TypeError	Wrong data type used in an operation or function	Try-except block
ValueError	Invalid value used with the correct data type	Try-except block
ZeroDivisionError	Division by zero	Try-except block
MyCustomError	User-defined cause	Try-except block

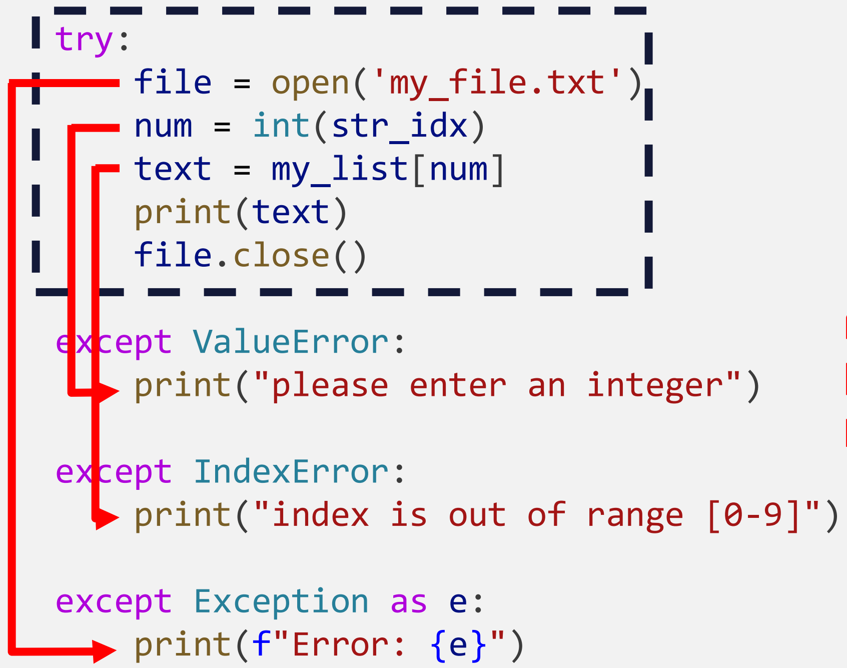
Single Exception



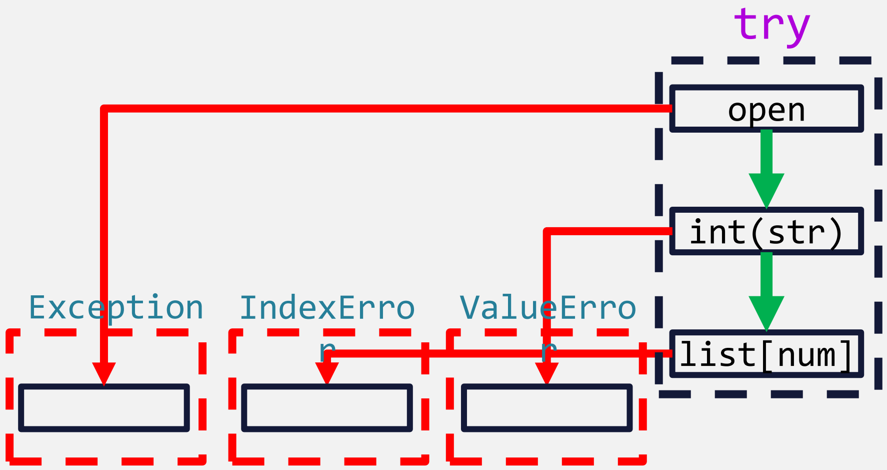
```
try:
    idx = int(str_idx)
    print('after conversion:', idx)
except ValueError:
    print('please enter integer!!!')
```



Multiple Exceptions



```
try:
    file = open('my_file.txt')
    num = int(str_idx)
    text = my_list[num]
    print(text)
    file.close()

except ValueError:
    print("please enter an integer")

except IndexError:
    print("index is out of range [0-9]")

except Exception as e:
    print(f"Error: {e}")
```



How to read errors?

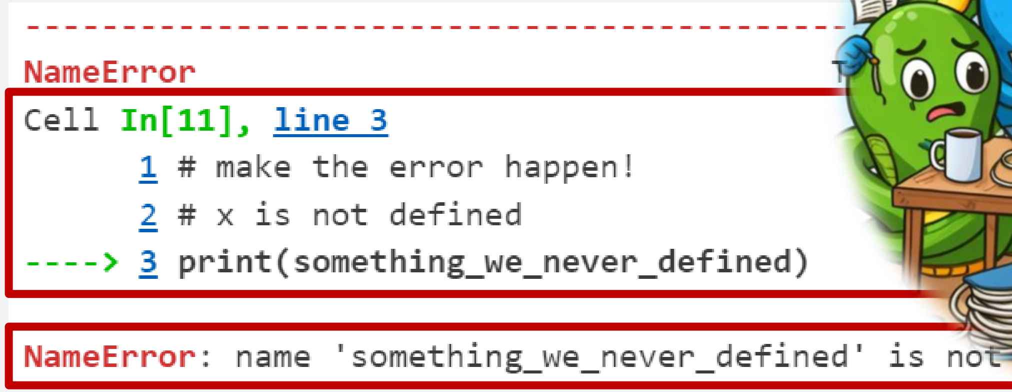
Errors are your friends

- Errors are inevitable. If you don't encounter them, then, you have no friends.
- Errors are not bad. They are a great way to learn and improve your programming skills.

Error Name

Error Location

Error Message



```
---------------------------------------------
NameError                                   T
Cell In[11], line 3
      1 # make the error happen!
      2 # x is not defined
----> 3 print(something_we_never_defined)

NameError: name 'something_we_never_defined' is not
```

Read complex errors

Simple steps to read:

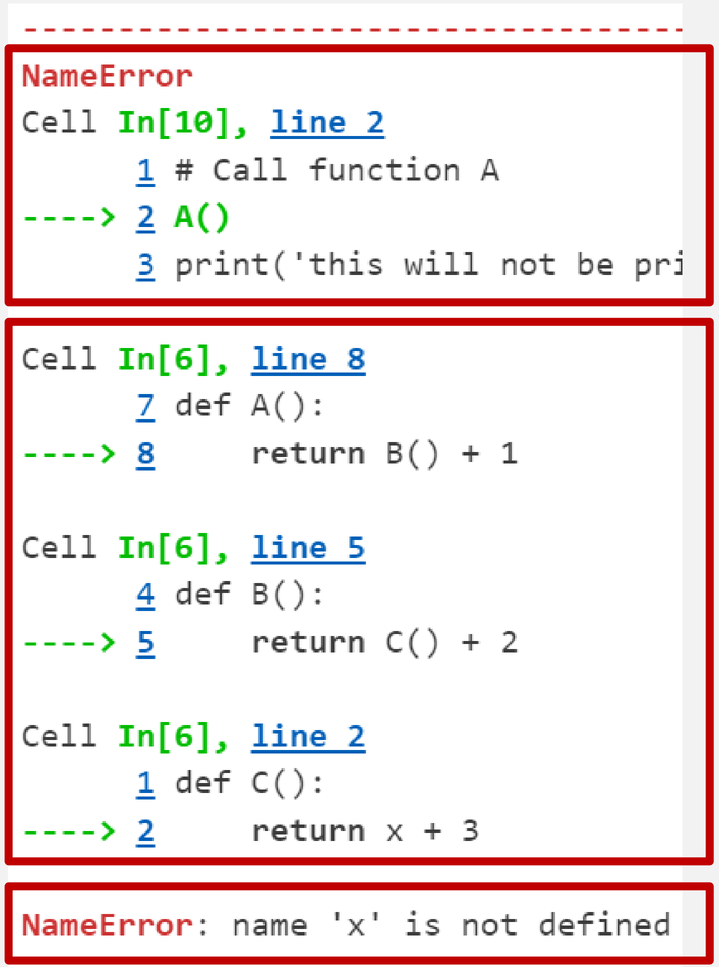
1. Read the bottom
2. Read from the top, down

```python
def C():
    return x + 3

def B():
    return C() +
2

def A():
    return B() +
1
A()
```

Line Causing the Error 2

Stack Trace

Error Message 1



```
---------------------------------------------------------------------------
NameError
Cell In[10], line 2
      1 # Call function A
----> 2 A()
      3 print('this will not be pri

Cell In[6], line 8
      7 def A():
----> 8     return B() + 1

Cell In[6], line 5
      4 def B():
----> 5     return C() + 2

Cell In[6], line 2
      1 def C():
----> 2     return x + 3

NameError: name 'x' is not defined
```

Thank you

SDAIA
الهيئة السعودية للبيانات والذكاء الاصطناعي
Saudi Data & AI Authority