Exception Handling

Agenda

What is Exception Handling?

What is Exception?

- Even if our code is syntactically correct, it may cause an error during execution.
- Errors detected during execution are called **exceptions.**
- Exception disrupts the normal flow of our code.

```
x = 20
y = 0
print(x//y) #This line is syntactically correct but we
              cannot divide any number by 0.
```

- x//y will raise an exception and our program terminates abruptly.
- These situations can be handled smoothly by including **exception handling code.**

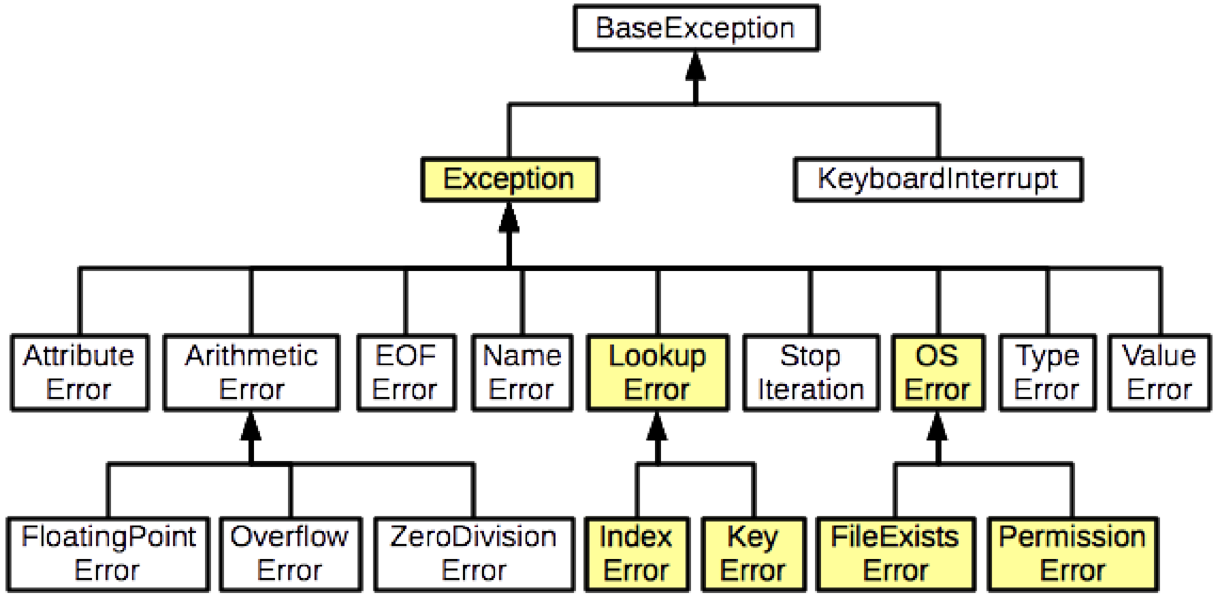
What is Exception Handling?

- **Exception handling** is the process of responding to exceptions when a program raises one.
- Exceptions include a user providing an invalid input, a file system error being encountered when trying to read or write a file, or a program attempting to divide by zero.
- Exception handling attempts to gracefully handle these situations so that a program (or worse, an entire system) does not crash.
- Exception handling can be included in our code using try and except blocks.
- We will learn about **try, except, else and finally** blocks shortly.

Built in Exceptions

Built in Exceptions

- Python has predefined built in exception classes to handle certain exceptions.
- All the built in exception classes are an instance of the class **BaseException.**



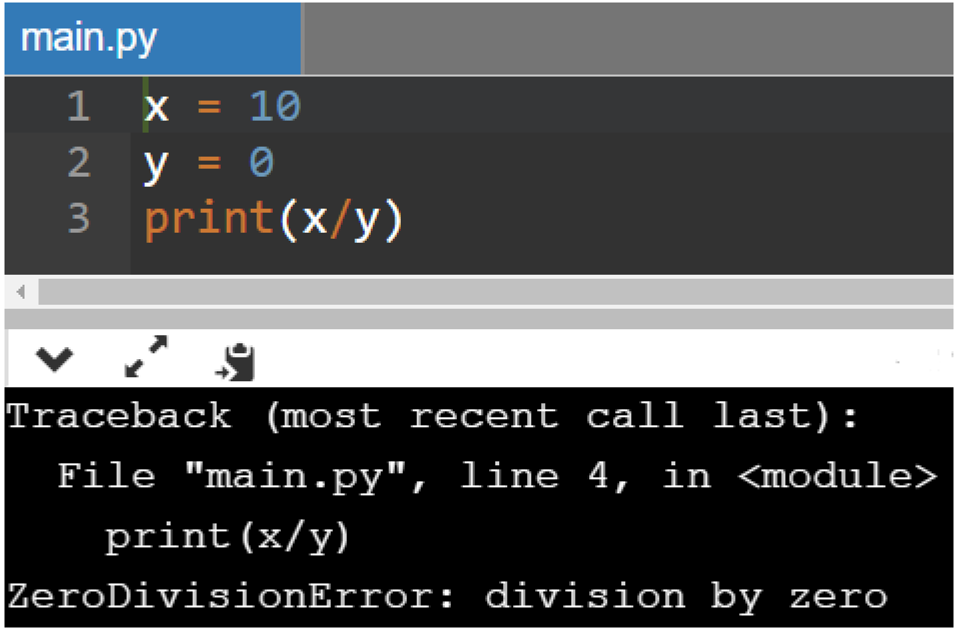
Built in Exceptions

Some important built in exceptions:

Exception class	Scenario
ZeroDivisionError	Raised when division or modulo by zero takes place for all numeric types.
SyntaxError	Raised when there is an error in Python syntax.
NameError	Raised when an identifier is not defined.
KeyError	Raised when the specified key is not found in the dictionary.
IndexError	Raised when an index is not found in a sequence.
ImportError	Raised when an import statement fails.
TypeError	Raised when an operation is attempted that is invalid for the specified data type.
ValueError	Raised when invalid (wrong data type) values are passed as arguments to the built-in functions.

ZeroDivisionError : Example

- Raised when division or modulo by zero takes place for all numeric types.

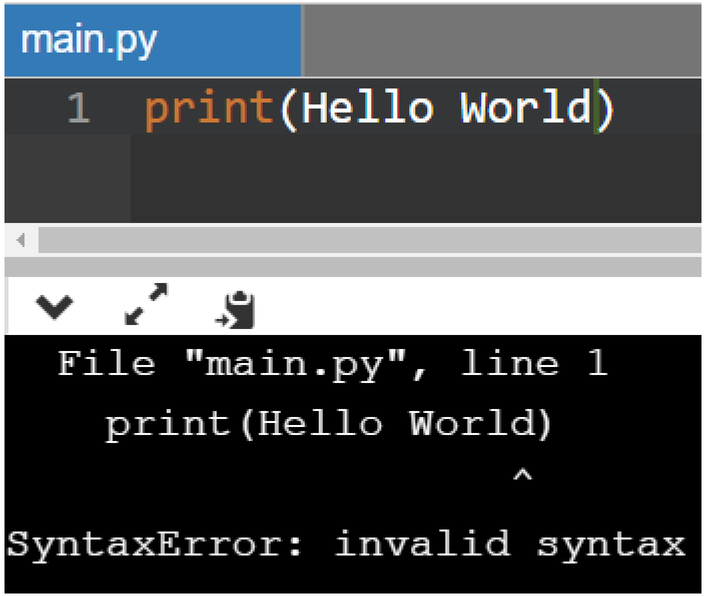

main.py

```
x = 10
y = 0
print(x/y)
```

```
Traceback (most recent call last):
  File "main.py", line 4, in <module>
    print(x/y)
ZeroDivisionError: division by zero
```

SyntaxError : Example

- Raised when there is an error in Python syntax.



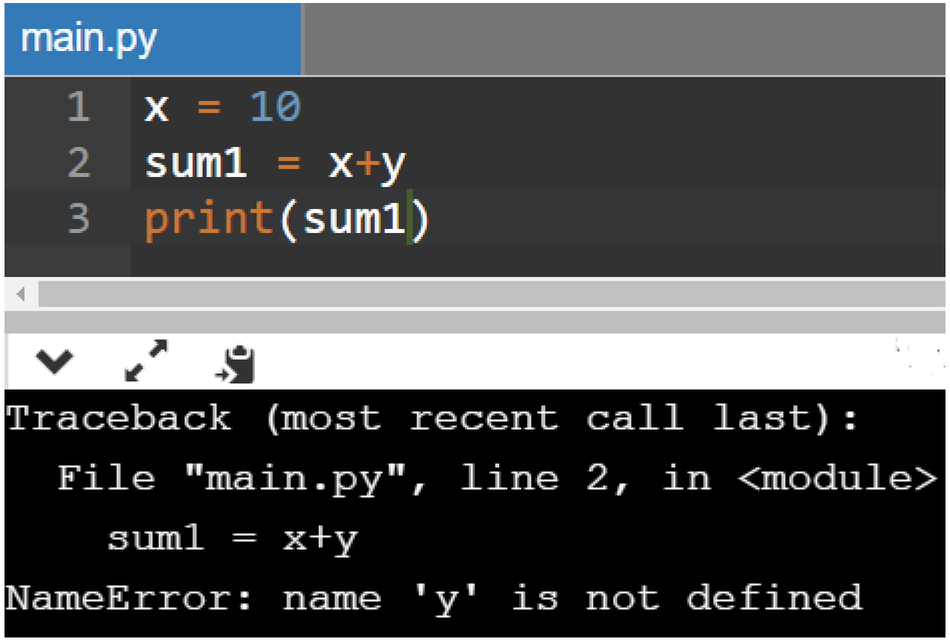
main.py

```
print(Hello World)
```

```
  File "main.py", line 1
    print(Hello World)
                    ^
SyntaxError: invalid syntax
```

NameError : Example

- Raised when an identifier is not defined.

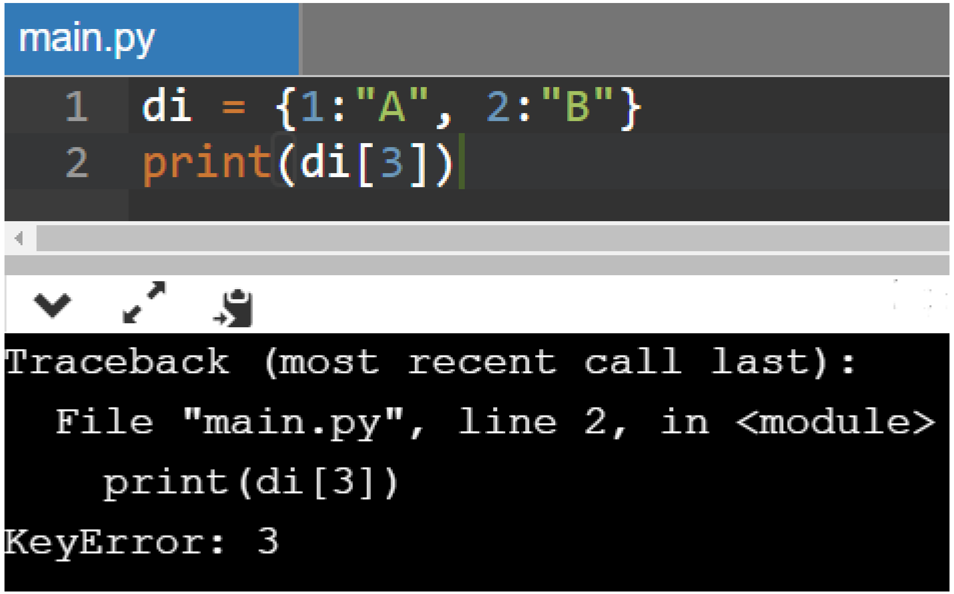

main.py

```python
x = 10
sum1 = x+y
print(sum1)
```

```
Traceback (most recent call last):
  File "main.py", line 2, in <module>
    sum1 = x+y
NameError: name 'y' is not defined
```

KeyError : Example

- Raised when the specified key is not found in the dictionary.

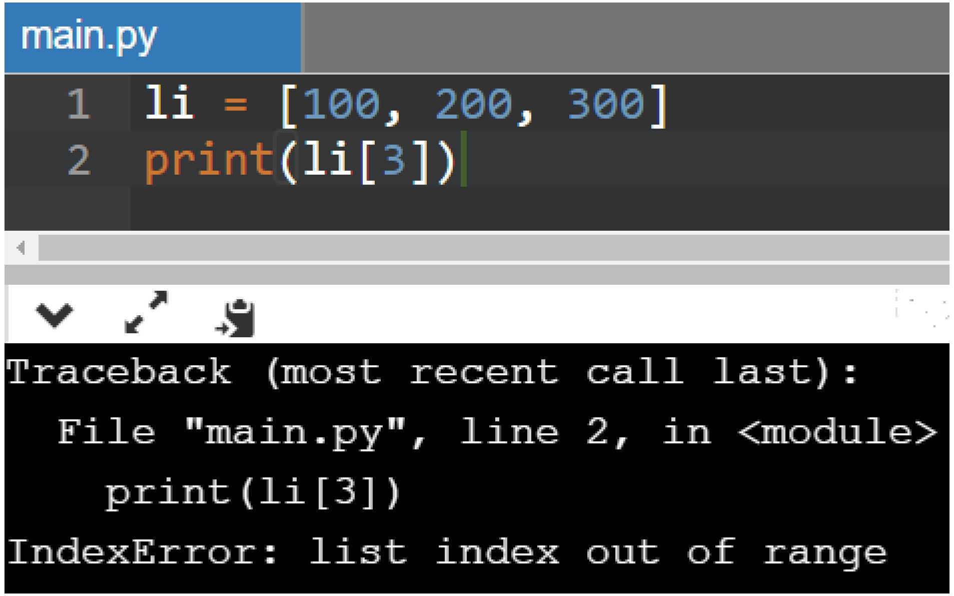


```python
di = {1:"A", 2:"B"}
print(di[3])
```

```
Traceback (most recent call last):
  File "main.py", line 2, in <module>
    print(di[3])
KeyError: 3
```

IndexError : Example

- Raised when an index is not found in a sequence.

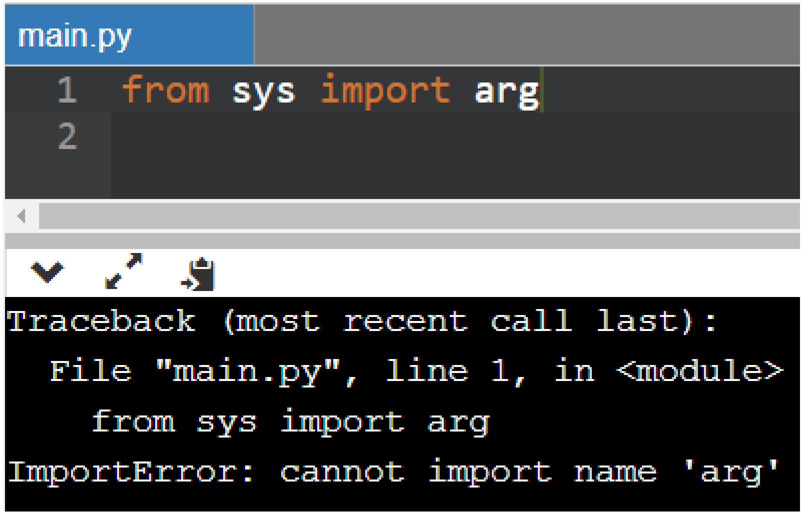


```
li = [100, 200, 300]
print(li[3])
```

```
Traceback (most recent call last):
  File "main.py", line 2, in <module>
    print(li[3])
IndexError: list index out of range
```

ImportError : Example

- Raised when an import statement fails.
- There is no such variable or function called arg present in the sys module.

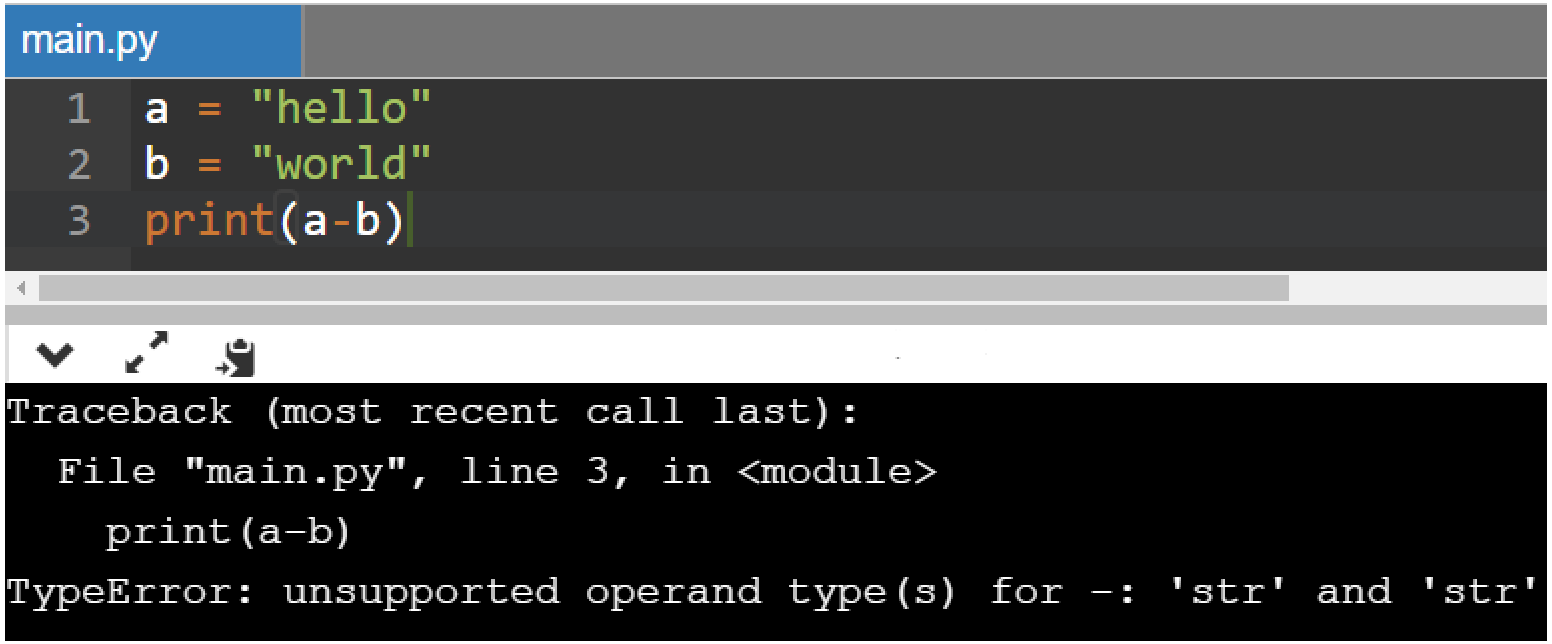


```
from sys import arg
```

```
Traceback (most recent call last):
  File "main.py", line 1, in <module>
    from sys import arg
ImportError: cannot import name 'arg'
```

TypeError : Example

- Raised when an operation is attempted that is invalid for the specified data type.

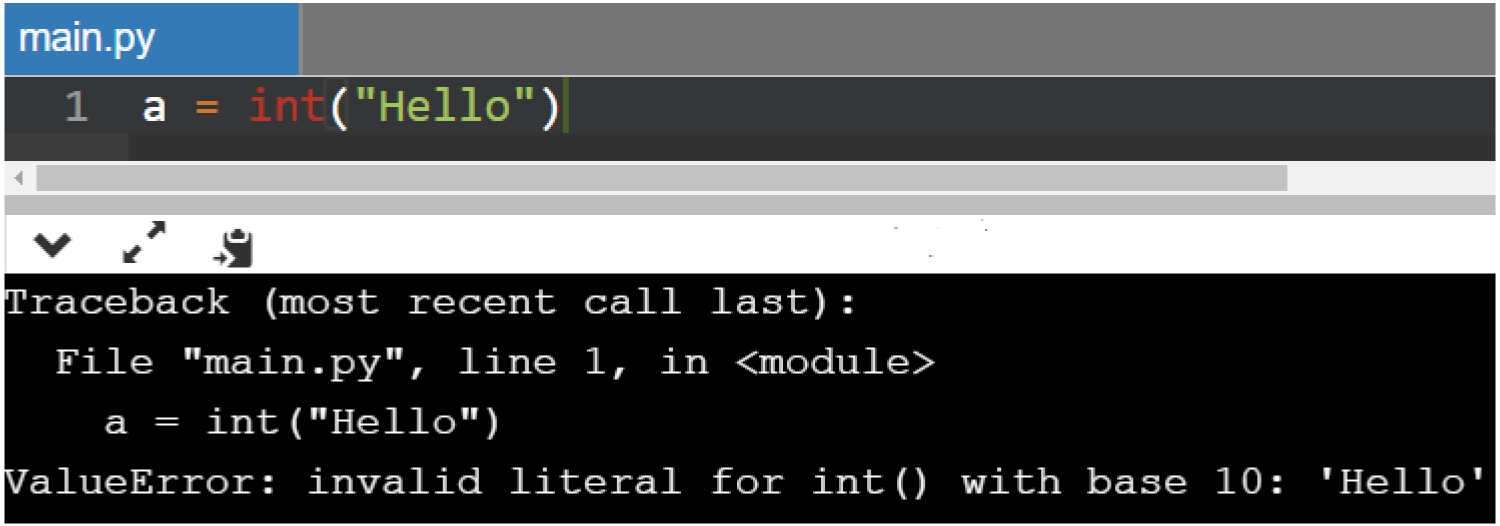

main.py

```python
a = "hello"
b = "world"
print(a-b)
```

```
Traceback (most recent call last):
  File "main.py", line 3, in <module>
    print(a-b)
TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

ValueError : Example

- Raised when invalid (wrong data type) values are passed as arguments to the built-in functions.


main.py

```python
a = int("Hello")
```

```
Traceback (most recent call last):
  File "main.py", line 1, in <module>
    a = int("Hello")
ValueError: invalid literal for int() with base 10: 'Hello'
```

Try and Except

Try and Except

- **try** block: Suspicious code which may raise an exception will be placed here.
- **except** block: Code to handle the exceptions will be placed here.

```
Basic syntax

try:
    statement-1
    .           .
    statement-n
except:
    statement-1
    .           .
    statement-n
```

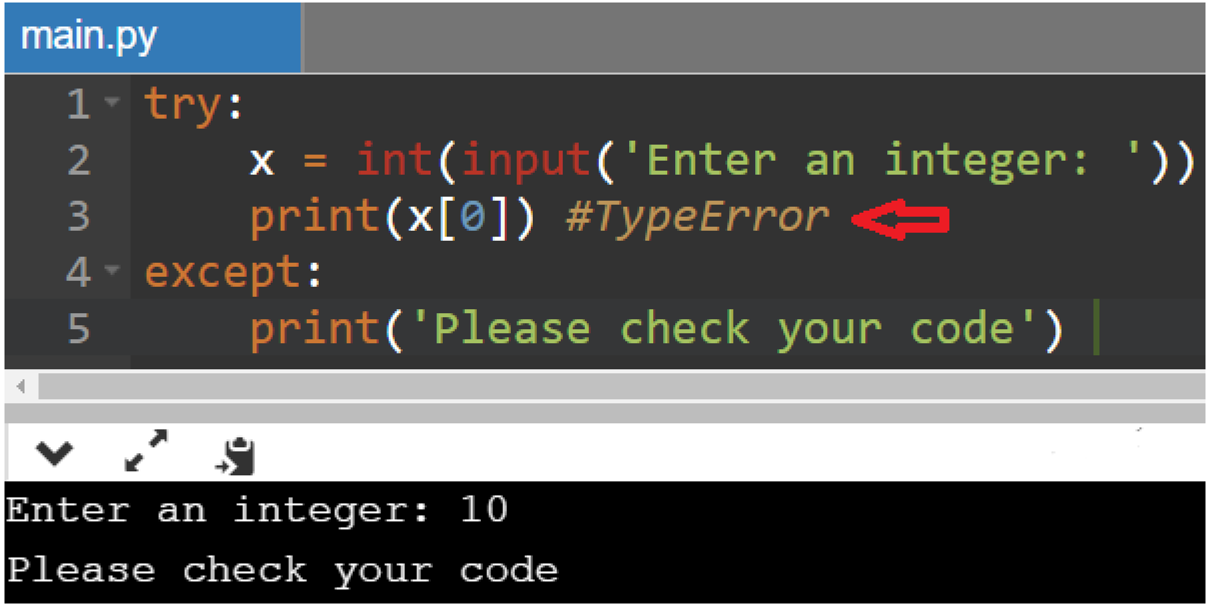
```
Basic syntax

try:
    statement-1
    .           .
    statement-n
except ExceptionClassName:
    statement-1
    .           .
    statement-n
```

Except without any exception class name

- **except** keyword without mentioning any specific exception class will catch any exception that is raised by the program.

Handling TypeError:

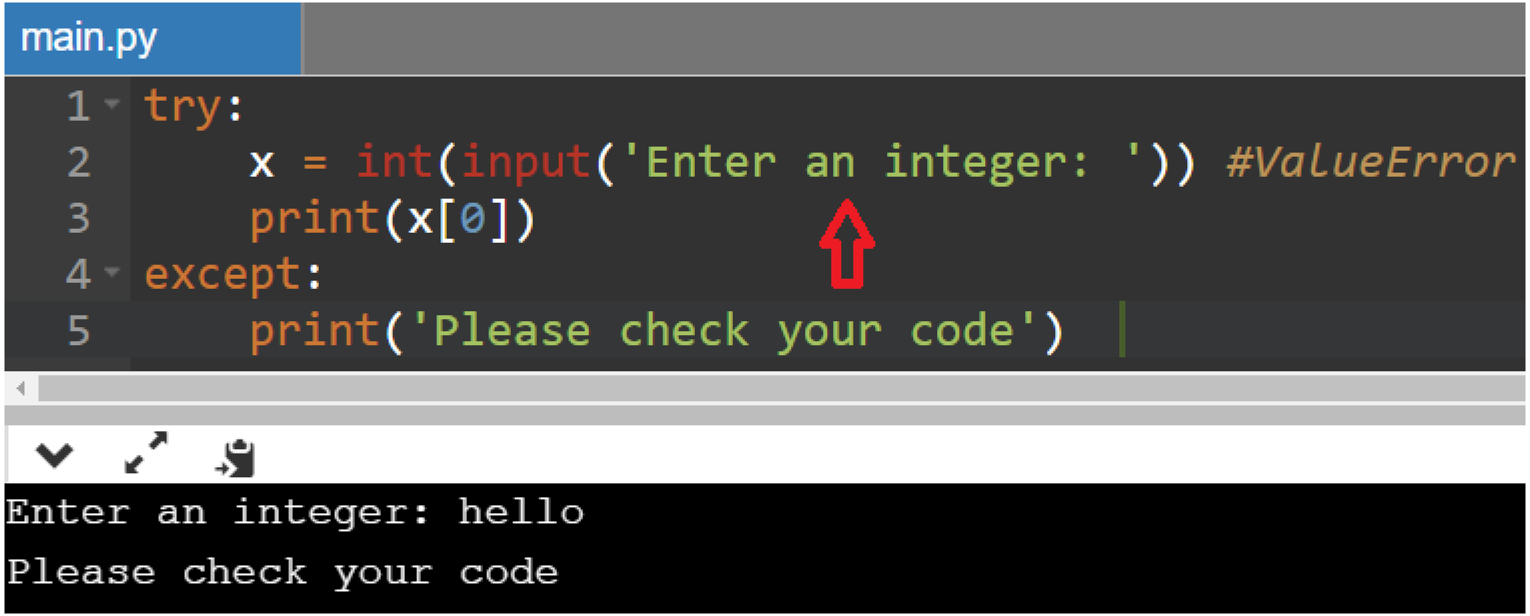


```python
try:
    x = int(input('Enter an integer: '))
    print(x[0]) #TypeError
except:
    print('Please check your code')
```

```
Enter an integer: 10
Please check your code
```

Except without any exception class name

Handling ValueError:


main.py

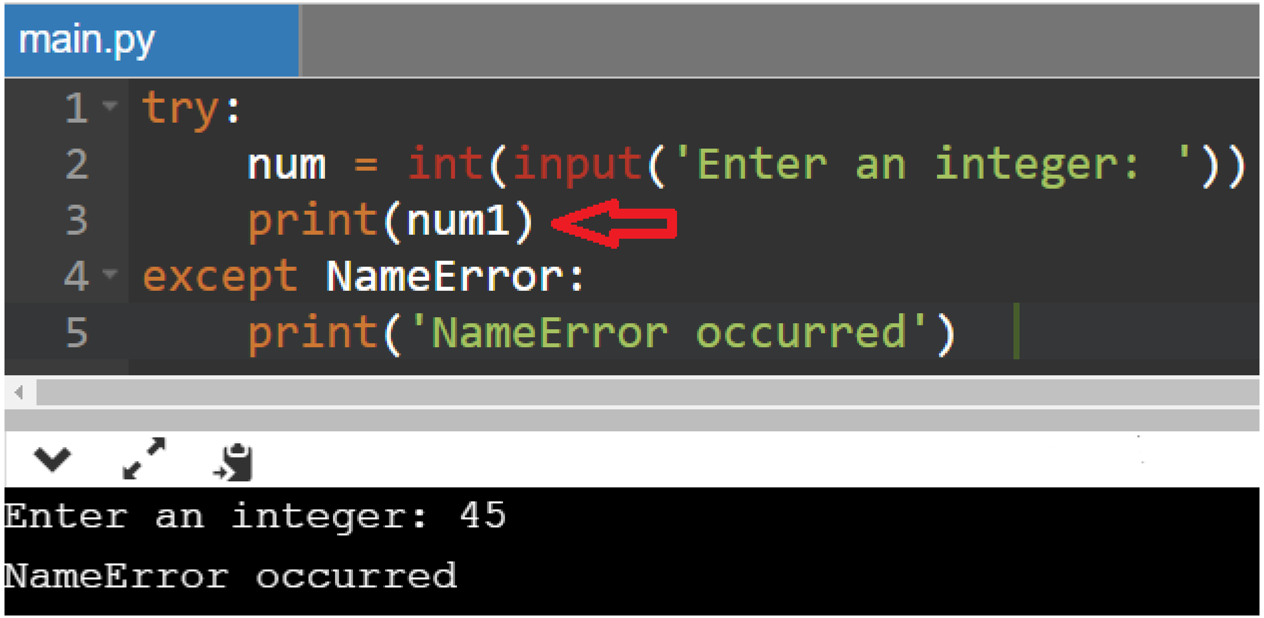
```
try:
    x = int(input('Enter an integer: ')) #ValueError
    print(x[0])
except:
    print('Please check your code')
```

```
Enter an integer: hello
Please check your code
```

Except with one exception class name

- **except** keyword with specific exception class name will catch and handle only that exception.

Handling NameError:



main.py

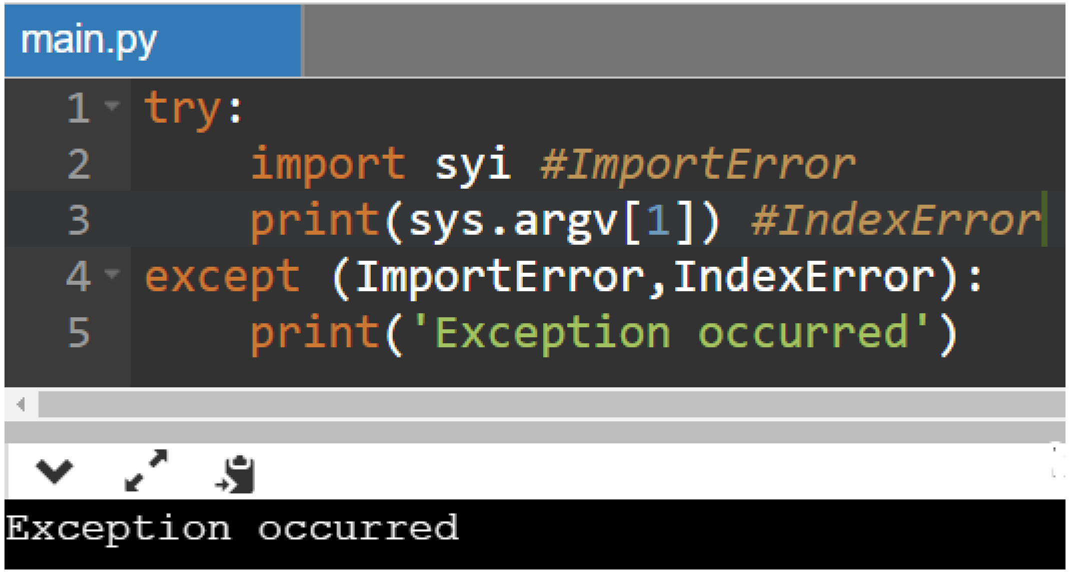
```python
try:
	num = int(input('Enter an integer: '))
	print(num1)
except NameError:
	print('NameError occurred')
```

```
Enter an integer: 45
NameError occurred
```

Except with multiple exception class names

- **except** keyword with more than one exception class name will catch and handle only those exceptions.

Handling only ImportError and IndexError:


main.py

```python
try:
	import syi #ImportError
	print(sys.argv[1]) #IndexError
except (ImportError,IndexError):
	print('Exception occurred')
```

```
Exception occurred
```

Try with multiple except blocks

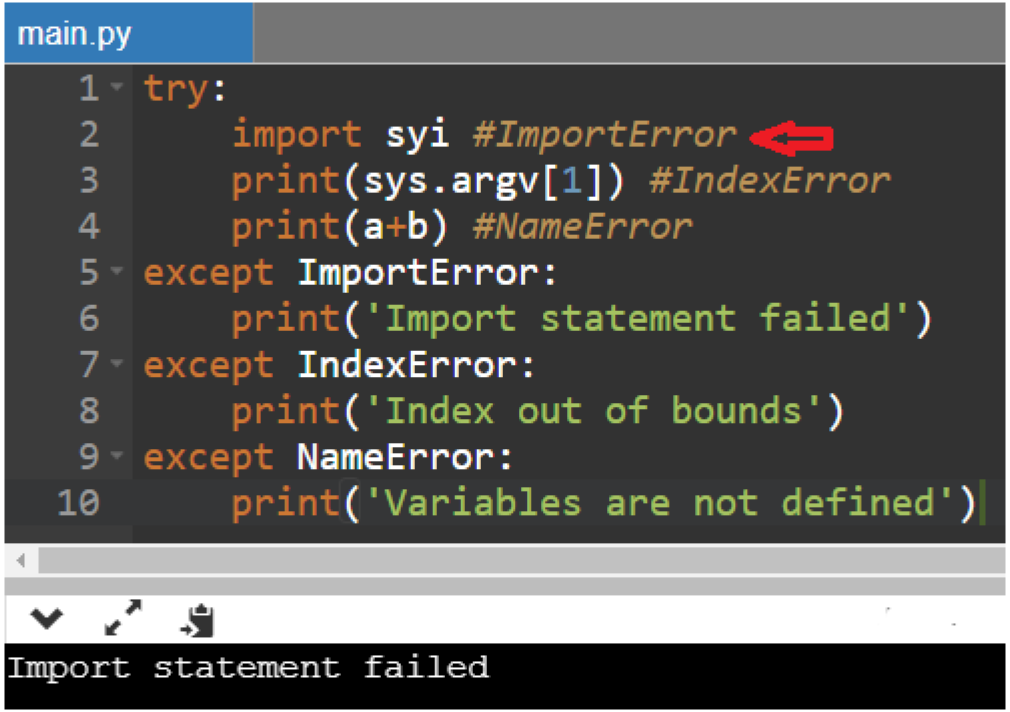
- We can have separate except blocks for every possible exception that can occur in our code.
- When exception occurs control goes to the respective exception block.

```
Basic syntax

try:
    statements
except ExceptionClassName1:
    statements
except ExceptionClassName2:
    statements
```

Try with multiple except blocks : Example

ImportError occurs:



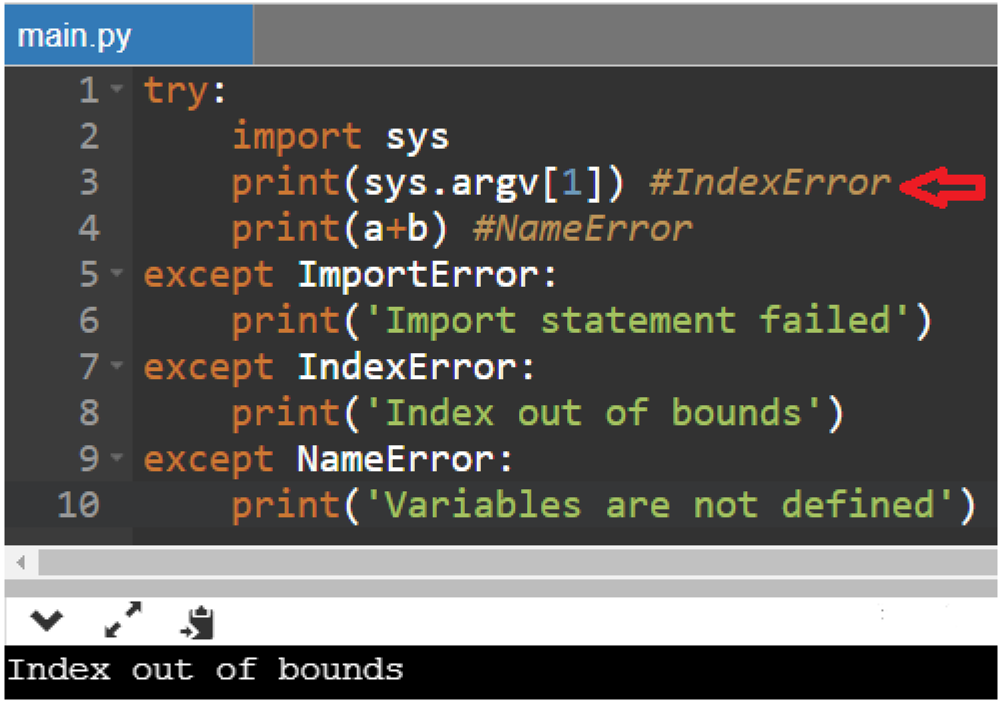
main.py

```python
try:
	import syi #ImportError
	print(sys.argv[1]) #IndexError
	print(a+b) #NameError
except ImportError:
	print('Import statement failed')
except IndexError:
	print('Index out of bounds')
except NameError:
	print('Variables are not defined')
```

```
Import statement failed
```

Try with multiple except blocks : Example

After correcting the import statement, IndexError occurs:



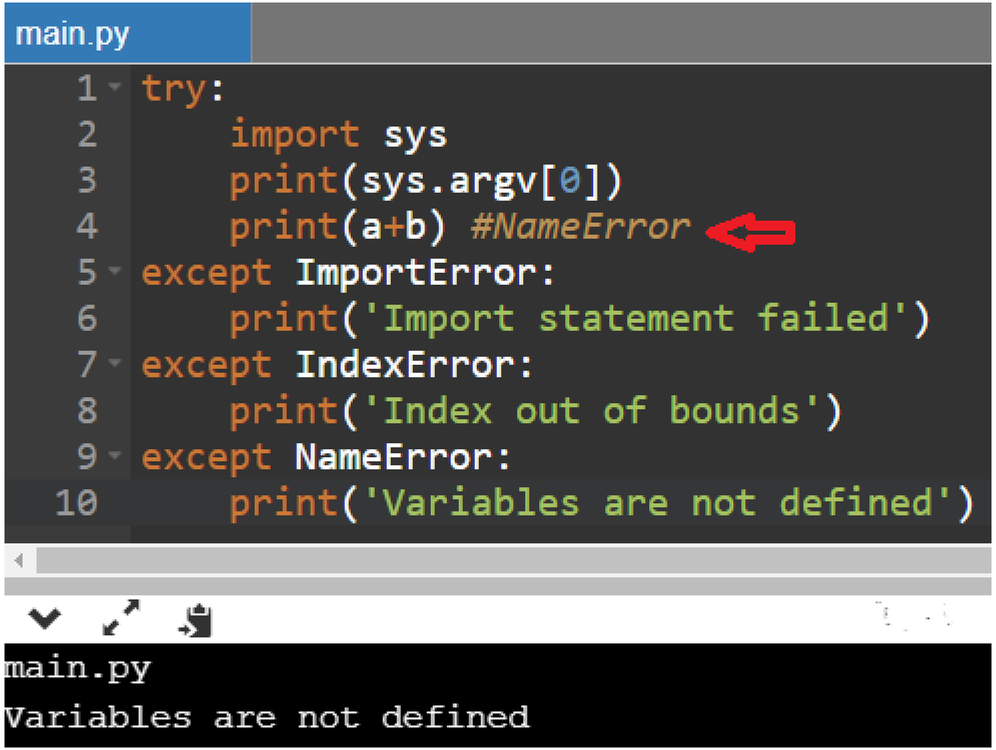
main.py

```python
try:
    import sys
    print(sys.argv[1]) #IndexError
    print(a+b) #NameError
except ImportError:
    print('Import statement failed')
except IndexError:
    print('Index out of bounds')
except NameError:
    print('Variables are not defined')
```

```
Index out of bounds
```

Try with multiple except blocks : Example

After correcting the index value, NameError occurs:



main.py

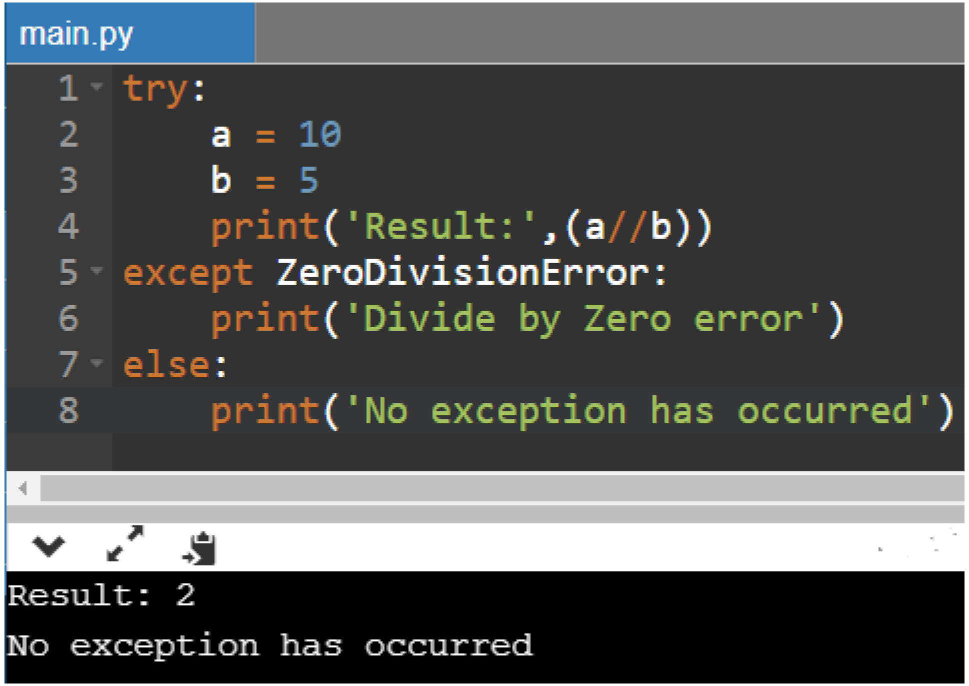
```
try:
    import sys
    print(sys.argv[0])
    print(a+b) #NameError
except ImportError:
    print('Import statement failed')
except IndexError:
    print('Index out of bounds')
except NameError:
    print('Variables are not defined')
```

```
main.py
Variables are not defined
```

Try, Except and Else

Try, Except and Else

- except block can be optionally followed by an **else** block.
- **else** block will be executed only when no exception has occurred.



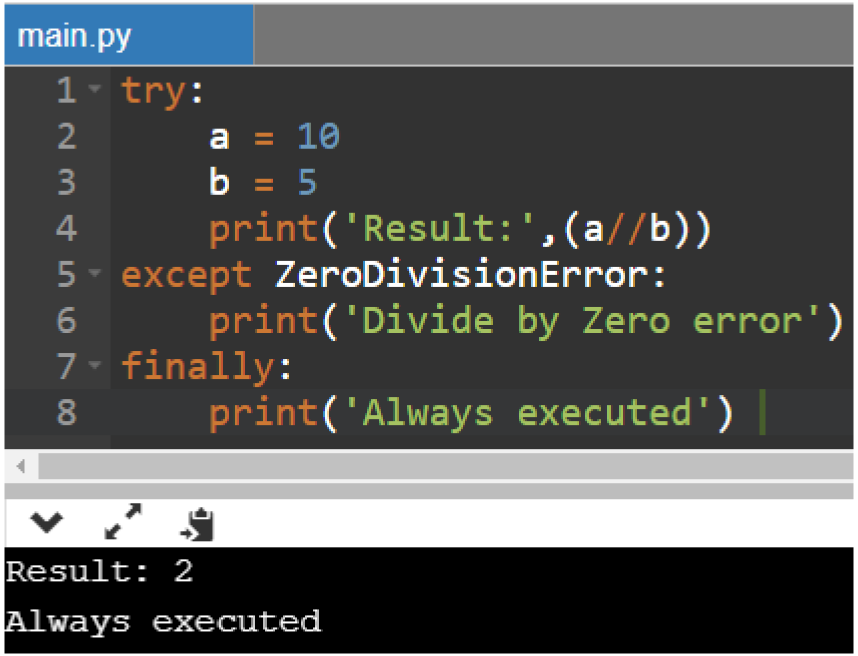
```python
try:
    a = 10
    b = 5
    print('Result:',(a//b))
except ZeroDivisionError:
    print('Divide by Zero error')
else:
    print('No exception has occurred')
```

```
Result: 2
No exception has occurred
```

Try, Except and Finally

Try, Except and Finally

- except block can be optionally followed by a **finally** block which will be executed always irrespective of whether any exception has occurred or not.



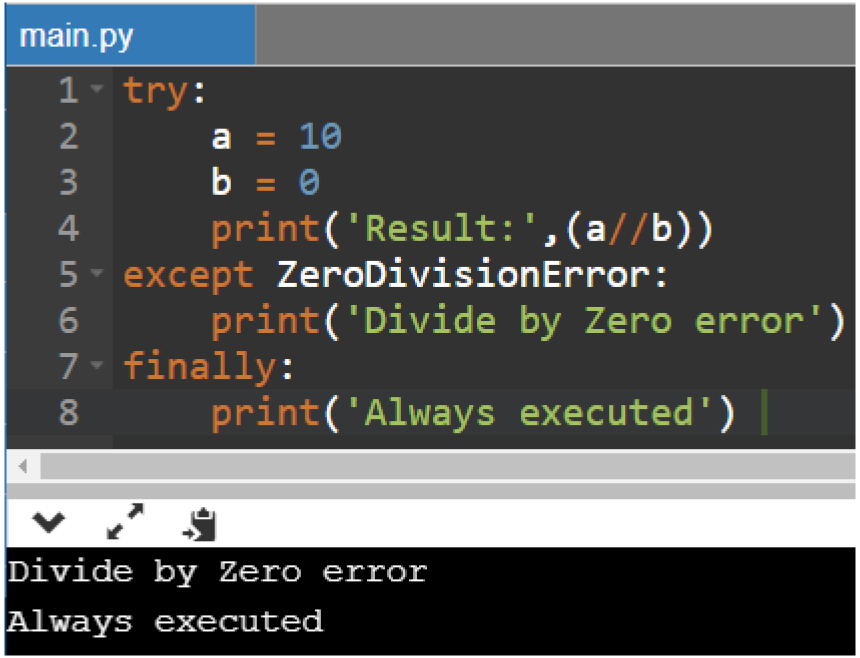
main.py

```python
try:
	a = 10
	b = 5
	print('Result:',(a//b))
except ZeroDivisionError:
	print('Divide by Zero error')
finally:
	print('Always executed')
```

```
Result: 2
Always executed
```

Try, Except and Finally

- Code to close the files or database connections at the end of our program execution can be included here.



```python
try:
    a = 10
    b = 0
    print('Result:',(a//b))
except ZeroDivisionError:
    print('Divide by Zero error')
finally:
    print('Always executed')
```

```
Divide by Zero error
Always executed
```

Thank you