Exceptions

CSCI3329 OOP in Python

Syntax error vs Run-time error

A syntax error happens when Python can't understand what you are saying.

A run-time error happens when Python understands what you are saying, but runs into trouble when following your instructions.

For example,

```
print("here")
print("heelo)
```

```
/home/dkim/PycharmProjects/CSCI3328/venv/bin/python /home/dkim/PycharmProjects/CSCI3328/temp.py
  File "/home/dkim/PycharmProjects/CSCI3328/temp.py", line 2
    print("heelo)
                ^
SyntaxError: EOL while scanning string literal

Process finished with exit code 1
```

Run-time errors

- Here are a few common run-time errors. Python is able to understand what the program says, but runs into problems when actually performing the instructions.
  - Using an undefined variable or function. This can also occur if you use capital letters inconsistently in a variable name:

```
print("here")
callMe = "Maybe"
print(callme)
```

```
/home/dkim/PycharmProjects/CSCI3328/venv/bin/python /home/dkim/PycharmProjects/CSCI3328/temp.py
here
Traceback (most recent call last):
  File "/home/dkim/PycharmProjects/CSCI3328/temp.py", line 4, in <module>
    print(callme)
NameError: name 'callme' is not defined

Process finished with exit code 1
```

Run-time errors

- Dividing by zero, which makes no sense in mathematics.

```
print("here")
print(1/0)
```

```
/home/dkim/PycharmProjects/CSCI3328/venv/bin/python /home/dkim/PycharmProjects/CSCI3328/temp.py
here
Traceback (most recent call last):
  File "/home/dkim/PycharmProjects/CSCI3328/temp.py", line 3, in <module>
    print(1/0)
ZeroDivisionError: division by zero

Process finished with exit code 1
```

Run-time errors

- Using operators on the wrong type of data

```
print("here")
print("you cannot add text and numbers" + 12)
```

```
/home/dkim/PycharmProjects/CSCI3328/venv/bin/python /home/dkim/PycharmProjects/CSCI3328/temp.py
here
Traceback (most recent call last):
  File "/home/dkim/PycharmProjects/CSCI3328/temp.py", line 3, in <module>
    print("you cannot add text and numbers" + 12)
TypeError: can only concatenate str (not "int") to str

Process finished with exit code 1
```

Run-time errors

Various errors can occur while programming with Python.

For example,

```
1    mylist = []
2    print(mylist[0])
```

```
/home/dkim/PycharmProjects/CSCI3328/venv/bin/python /home/dkim/PycharmProjects/CSCI3328/temp.py
Traceback (most recent call last):
  File "/home/dkim/PycharmProjects/CSCI3328/temp.py", line 2, in <module>
    print(mylist[0])
IndexError: list index out of range

Process finished with exit code 1
```

Run-time errors

Another example is here.

```
text = 'abc'
number = int(text)
```

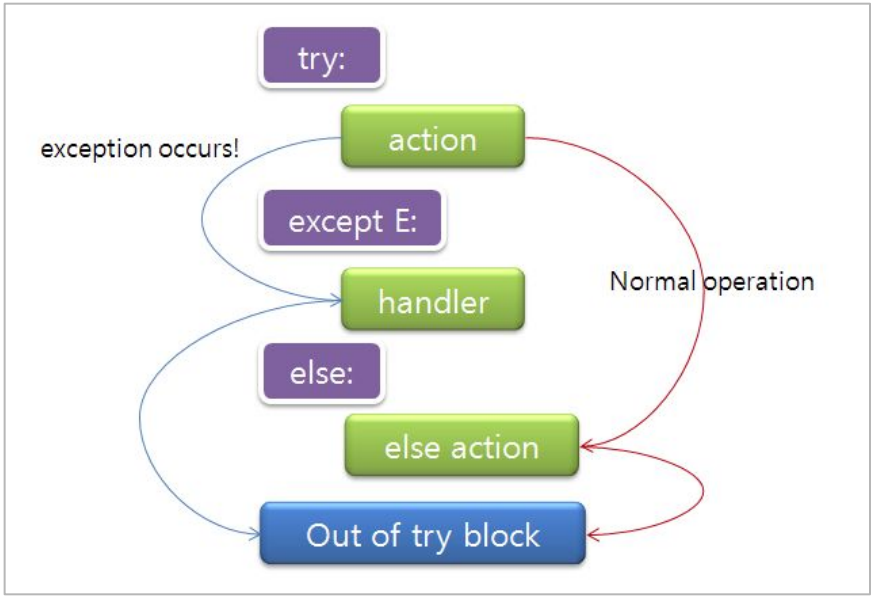
```
/home/dkim/PycharmProjects/CSCI3328/venv/bin/python /home/dkim/PycharmProjects/CSCI3328/temp.py
Traceback (most recent call last):
  File "/home/dkim/PycharmProjects/CSCI3328/temp.py", line 3, in <module>
    number = int(text)
ValueError: invalid literal for int() with base 10: 'abc'

Process finished with exit code 1
```

Exception

Exceptions in which this error occurs can be a flexible programming tool.

If you put the code that is likely to cause an error in `try` and write down the name of the error that may occur after the `except`, you can handle it separately when an error occurs without terminating the program.



try/except

For example,

```
text = 'abc'
try:
    number = int(text)
except ValueError:
    print("{} is not a number".format(text))
print("here")
```

```
/home/dkim/PycharmProjects/CSCI3328/venv/bin/python /home/dkim/PycharmProjects/CSCI3328/temp.py
abc is not a number
here

Process finished with exit code 0
```

What to do if you don't know the exception name

```
3    try:
4        mylist = []
5        print(mylist[0])
6    except:
7        print('Error')
```

```
/home/dkim/PycharmProjects/CSCI3328/venv/b
Error

Process finished with exit code 0
```

What to do if you don't know the exception name

```
3   try:
4       mylist = []
5       print(mylist[0])
6       text = 'abc'
7       number = int(text)
8   except Exception as ex:
9       print('Error: ', ex)
10  print("here")
```

```
/home/dkim/PycharmProjects/CSCI3328/venv/bin/pyt
Error:  list index out of range
here

Process finished with exit code 0
```

How to raise an error yourself-raise

- We can use `raise` to throw an exception if a condition occurs. The statement can be complemented with a custom exception.
  - User-directly generating errors
  - With heavy use, the code becomes difficult to read.
- For example, if you want to throw an error when a certain condition occurs using raise, you could go about it like this:

```
x = 10
if x > 5:
      raise Exception('x should not exceed 5. The value of x was: {}'.format(x))
```

```
Traceback (most recent call last):
  File "<input>", line 4, in <module>
Exception: x should not exceed 5. The value of x was: 10
```

The `AssertionError` Exception

Instead of waiting for a program to crash midway, you can also start by making an *`assert`* in Python.

We assert that a certain condition is met.

If this condition turns out to be **True**, then that is excellent! The program can continue.

If the condition turns out to be **False**, you can have the program throw an AssertionError exception.

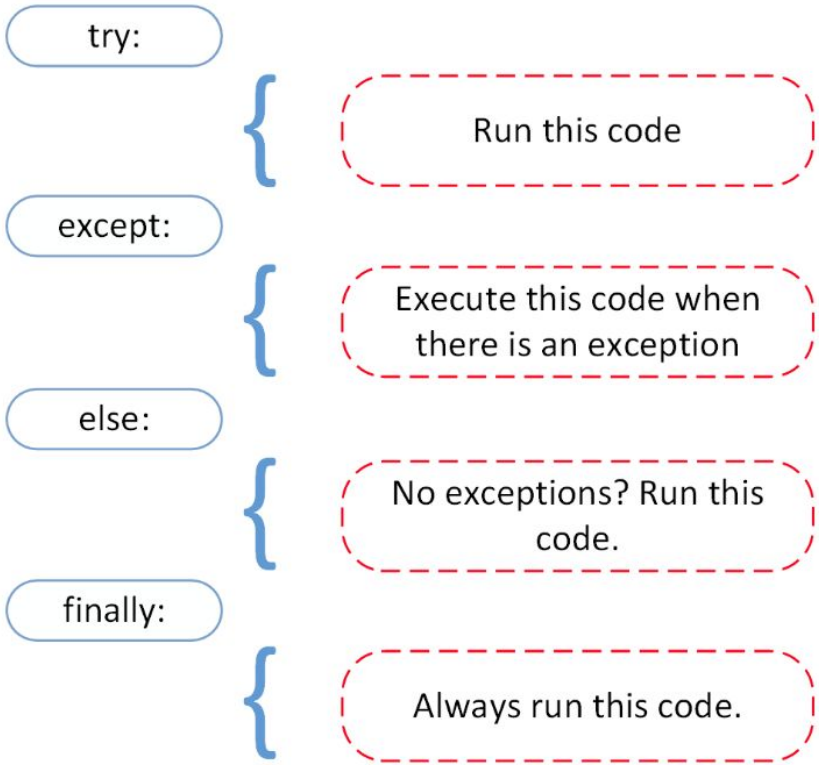
For example

```
import sys

print(sys.platform)

assert ('linux' in sys.platform), "This code runs on Linux only."
```

finally


try:
Run this code
except:
Execute this code when there is an exception
else:
No exceptions? Run this code.
finally:
Always run this code.

Example

```python
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("Error: Division by zero!")
    else:
        print("Division successful! Result:", result)
    finally:
        print("Execution completed.")

# Example usage:
divide(10, 2)
print()
divide(10, 0)
```

```
Division successful! Result: 5.0
Execution completed.

Error: Division by zero!
Execution completed.
```