Program Execution

Table of Contents

1 Introduction

How was the pre-class content? Did you cover the Program Execution?

Students, drag the icon!

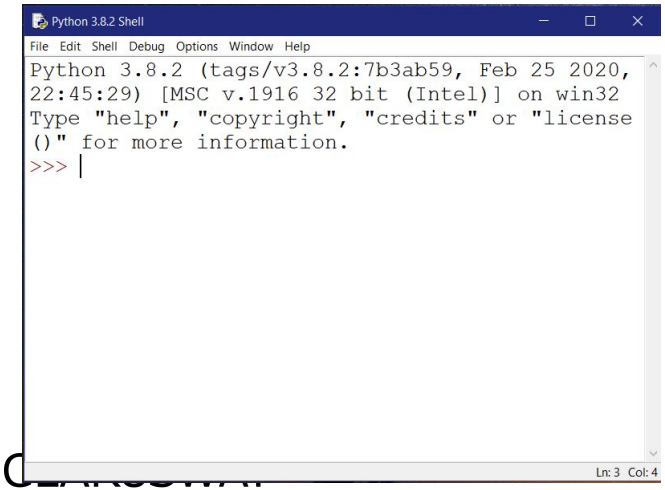
Introduction (review)

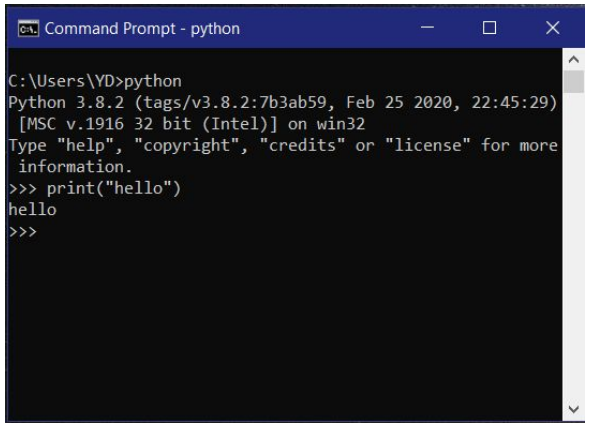
- Have you ever thought, after writing lines of Python code, how they do amazing things?
- We will try to understand how Python codes work, in other words, we will take a look at what's happening when you run the codes.
- Even if we run the simple conventional Python syntax : `print('Hello World!')`, very complex things are performed in the same way.

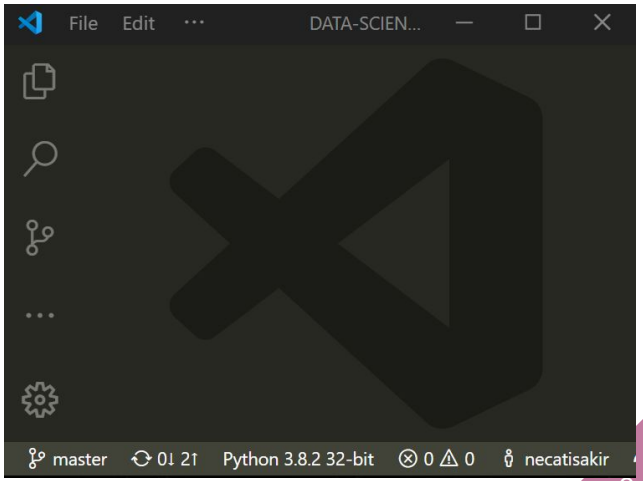
Introduction (review)

- The Python scripts (a file with **.py** extension) contain Python codes and we can run these files via several **interpreter** tools such as :

Python Shell | **OS Console** | **IDEs**







Introduction (review)

- Since the very beginning of this course, you have probably heard terms such as **interpreted** or **compiled** languages. And most likely, you learned that Python was an interpreted type of programming language. What does all this mean? Let's dive into the Pythonic world.
- The process of program execution (the program flow) basically looks like this :

2 The Interpretation

The Interpretation (review)

What is the interpretation process?

?

The Interpretation (review)

What is the interpretation process?

?

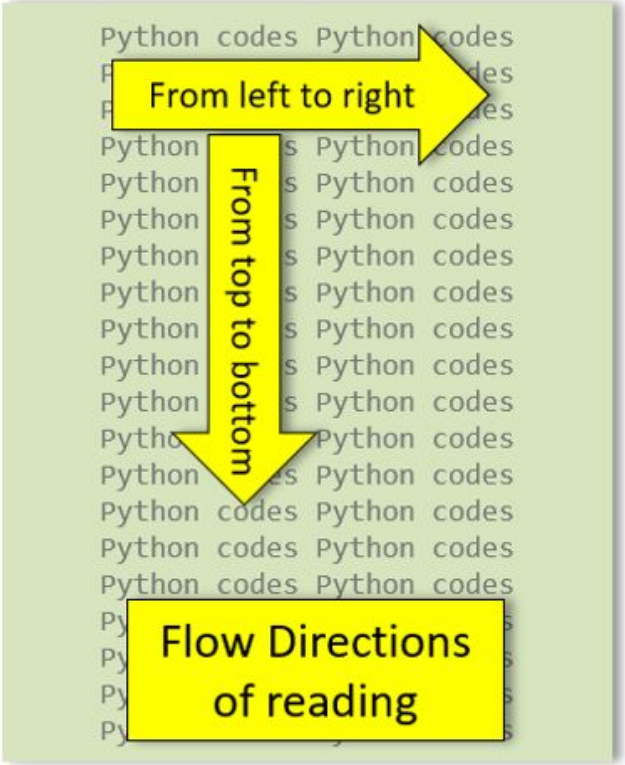
Simply, we can say that this is like **reading your codes.**

The Interpretation (review)

What is the interpretation process?

?

Simply, we can say that this is like **reading your codes.**



The Interpretation (review)

CPython

Python's **default interpreter** is a software written in the **C** programming language known as **CPython**.

- There are several other interpreters available. Let's take a look at these.

Could you remember these interpreters?

Write down their names.

Students, write your response!

The Interpretation (review)

Jython	It is an interpreter that works with a Java-based algorithm and converts Python codes into Java-compatible byte code, which will be executed later by the Java Virtual Machine.

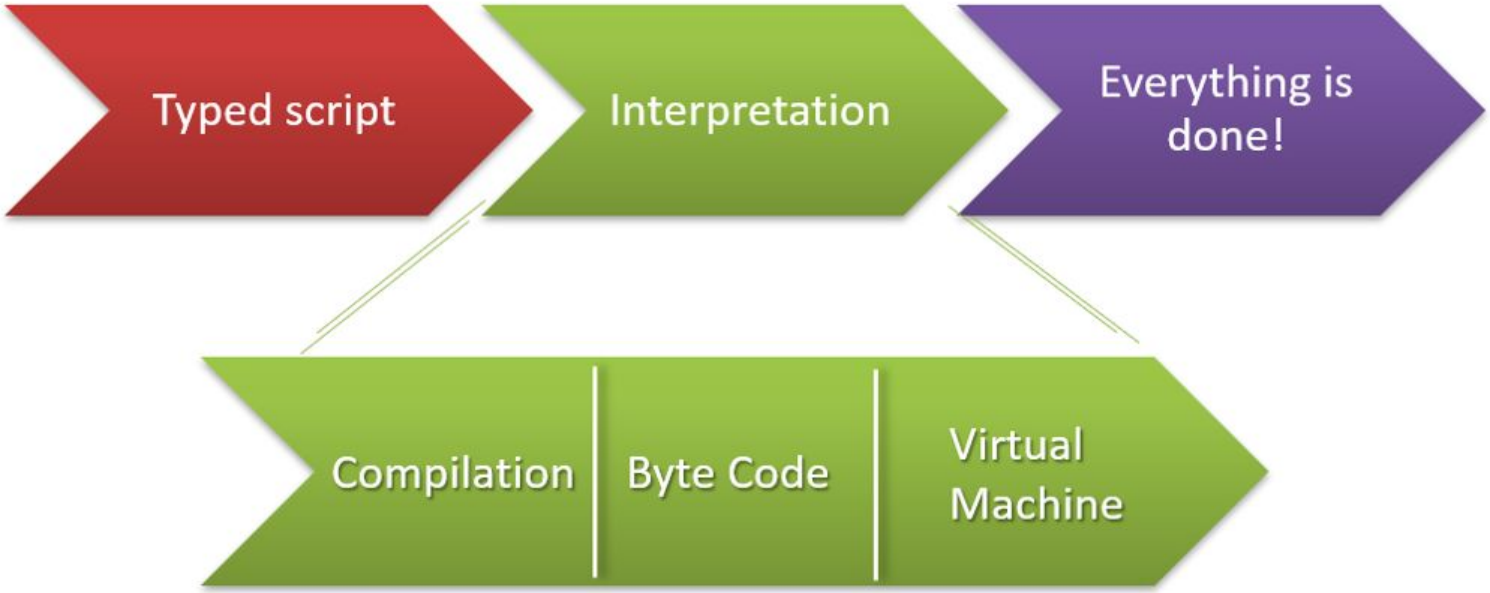
The Interpretation (review)

Jython	It is an interpreter that works with a Java-based algorithm and converts Python codes into Java-compatible byte code, which will be executed later by the Java Virtual Machine.
PyPy	It is a replacement for CPython. It is built using the RPython language that was co-developed with it. The main reason to use it instead of CPython is speed: RPython (Restricted Python) provides some restrictions to the usual Python code.

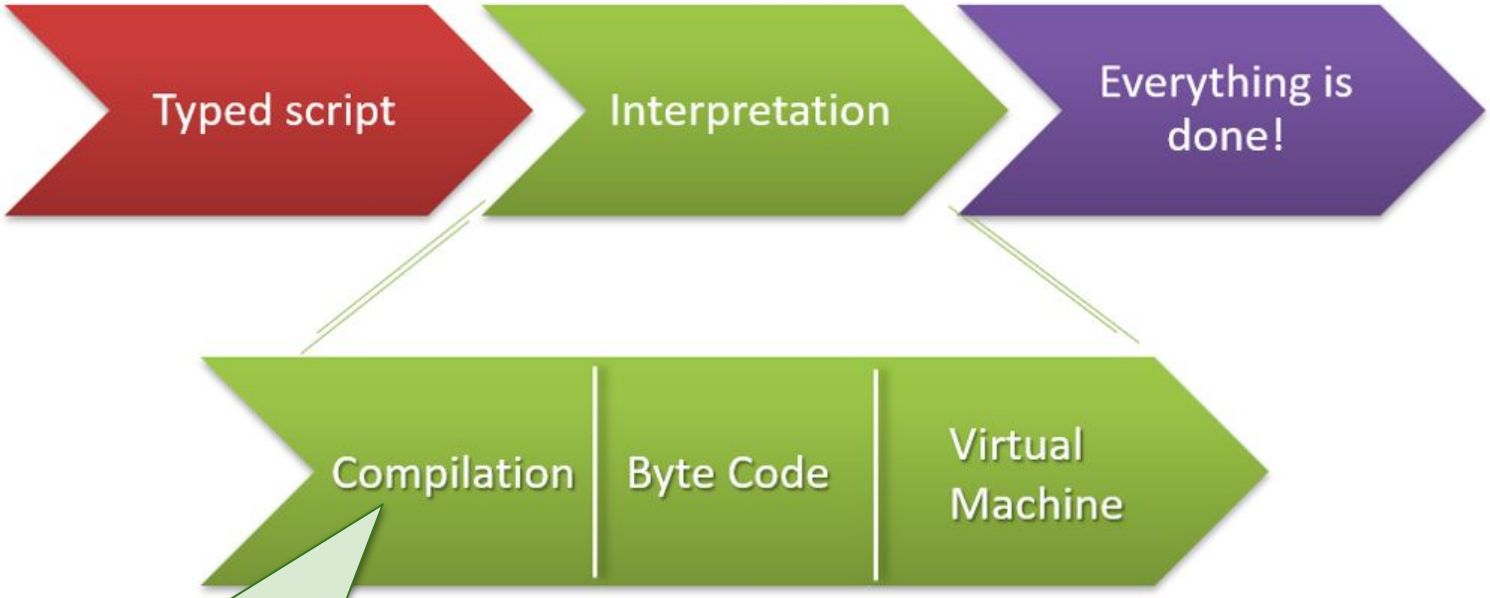
The Interpretation (review)

Jython	It is an interpreter that works with a Java-based algorithm and converts Python codes into Java-compatible byte code, which will be executed later by the Java Virtual Machine.
PyPy	It is a replacement for CPython. It is built using the RPython language that was co-developed with it. The main reason to use it instead of CPython is speed: RPython (Restricted Python) provides some restrictions to the usual Python code.
IronPython	It is an open-source implementation of the Python which is tightly integrated with the `.NET` Framework.

The Interpretation Process (re

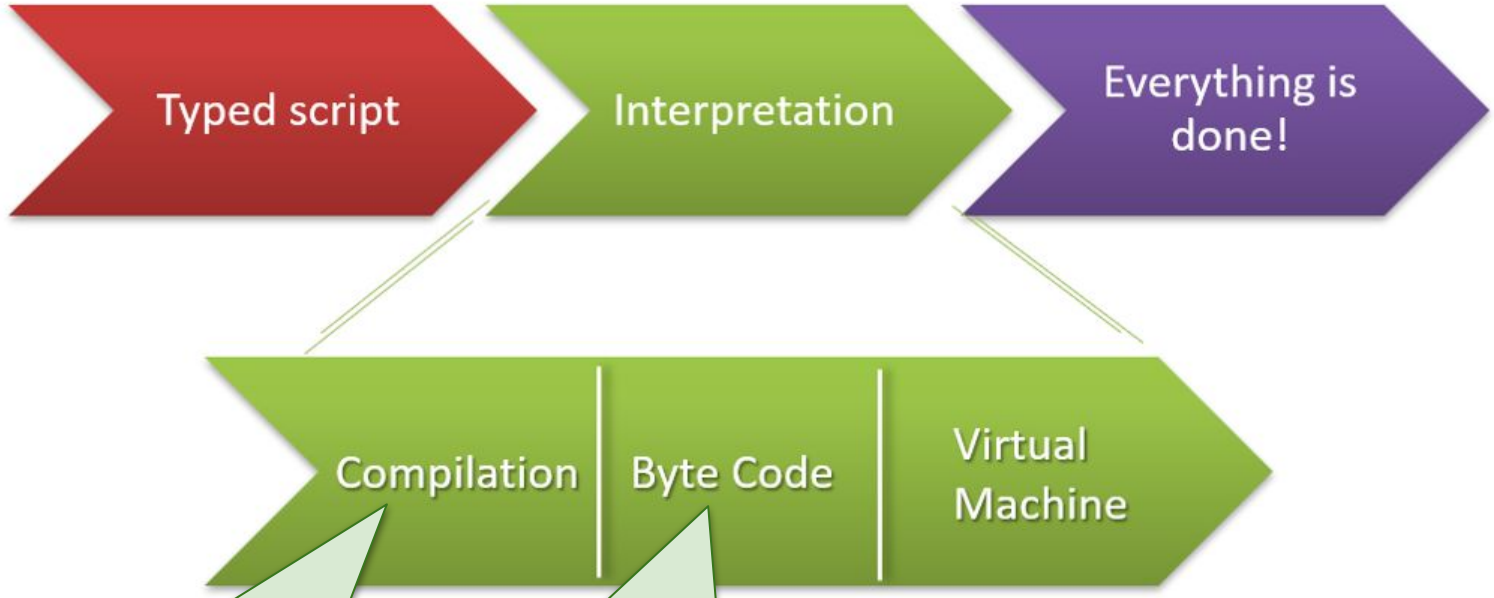


The Interpretation Process (re



The compiler translates your Python statements (source code) into **`byte-code`**.

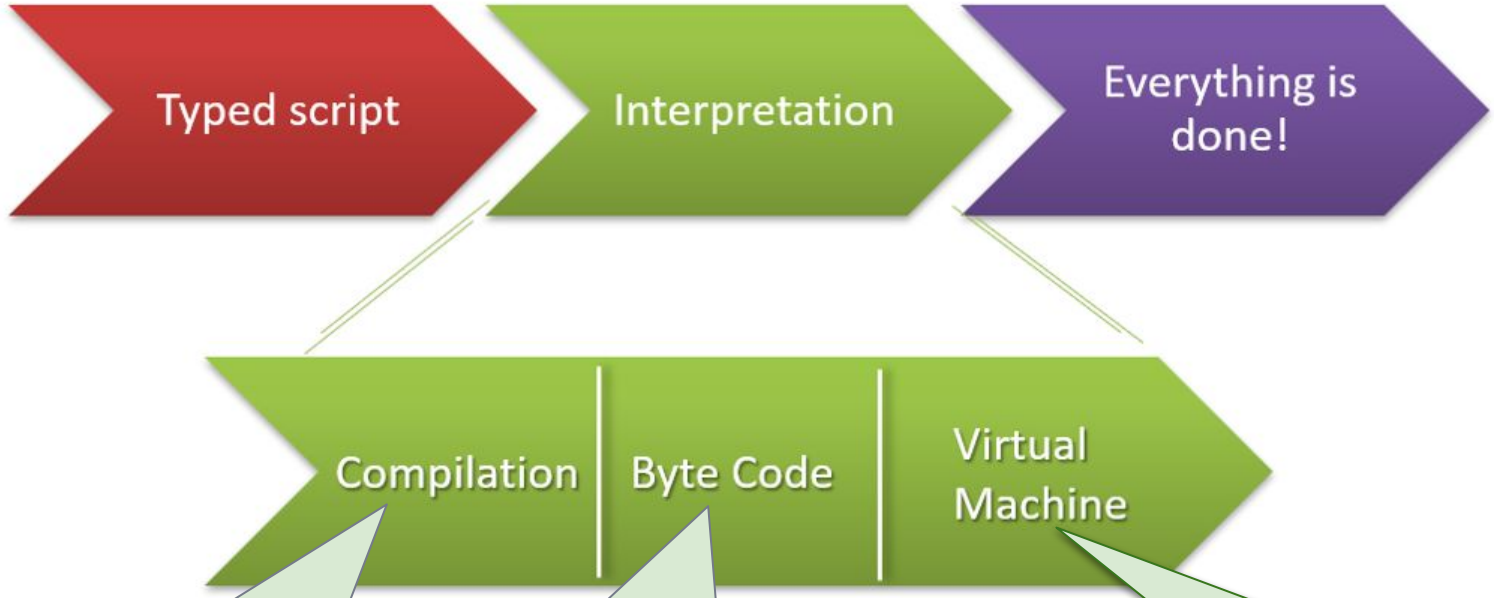
The Interpretation Process (re



The compiler translates your Python statements (source code) into **`byte-code`**.

Byte code is platform-independent, primitive level and specific version of the Python source code.

The Interpretation Process (re



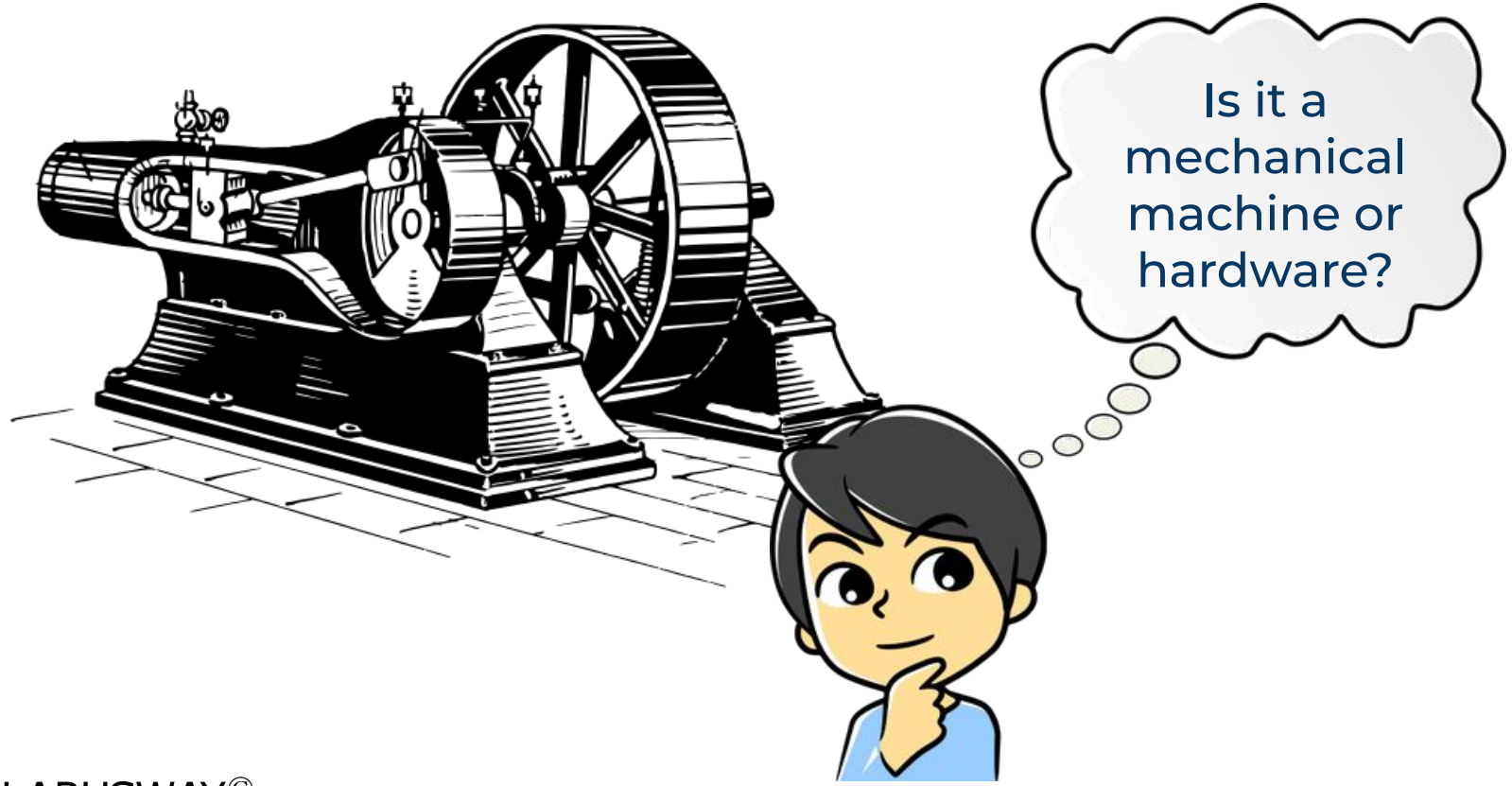
The compiler translates your Python statements (source code) into byte-code.

Byte code is platform-independent, primitive level and specific version of the Python source code.

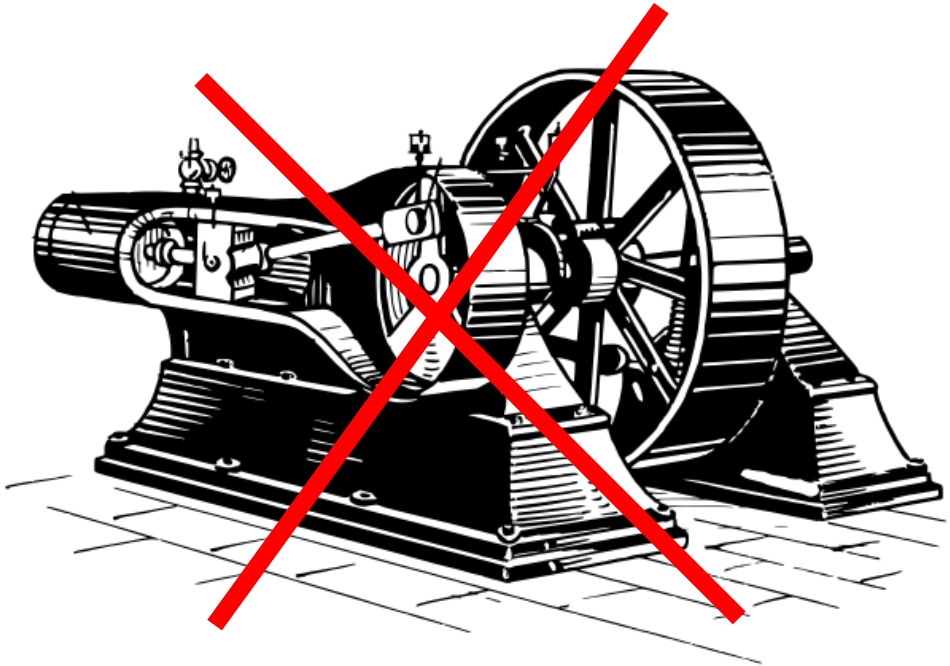
After compilation of Python statements into **byte-code**, it is now time for Python **Virtual Machine** to run.

3 Virtual Machine

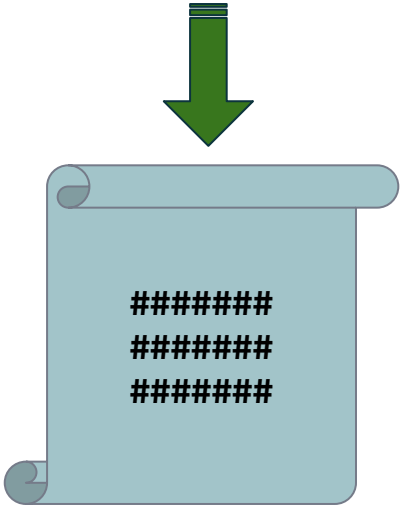
What is a Virtual Machine (review)



Virtual Machine (review)

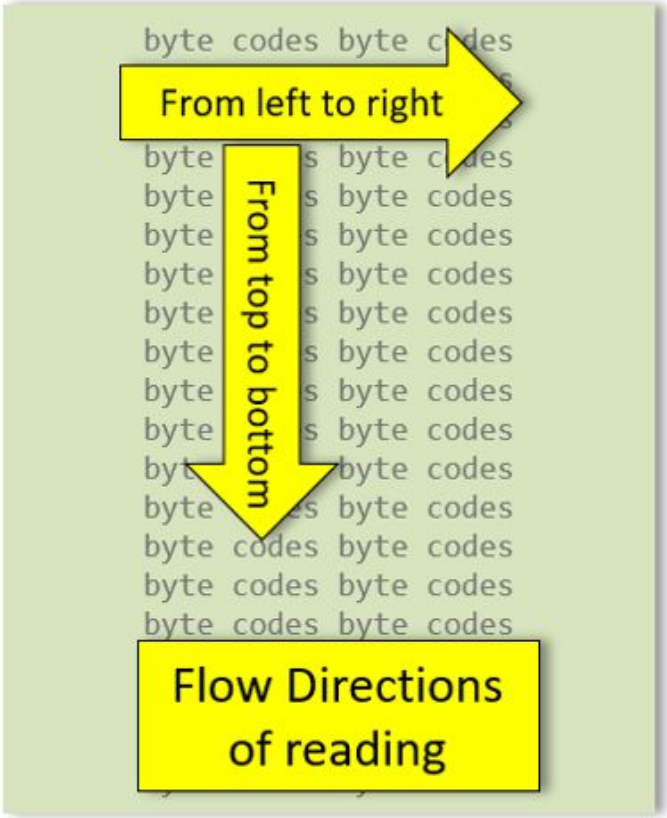


It is actually nothing but software made up of a large piece of code



Virtual Machine (review)

- Just like in compiler, Virtual Machine executes byte-codes that come into it by reading line by line from **top to bottom** and **from left to right**.
- It does not require a separate library or program installation. You do not need to know how the codes that go from the Python statements to the byte-code and from the byte-code to the Virtual Machine are created and executed.



How well did you like this lesson? Did you learn all topics?

Errors

Table of Contents

1 Introduction

What can you say about this lesson?

Summarize what you've learned from pre-class content.

Students, write your response!

Introduction (review)

- Sometimes you can see that the codes with the simplest syntax can give an error when you never expected.
- Consider the following example :

```
1  print("Don't say 'I never make a mistake'"
```

What is the output? Try to figure out in your mind...

Students, write your response!

Introduction (review)

- Sometimes you can see that the codes with the simplest syntax can give an error when you never expected.
- Consider the following example :

```
1  print("Don't say 'I never make a mistake'"
```

```
1  Traceback (most recent call last):
2  File "code.py", line 1
3      print("Don't say 'I never make a mistake'"
4                                                ^
5  SyntaxError: unexpected EOF while parsing
```

Introduction (review)

- In the error message appeared on the screen, there is the term **Traceback**.
- It is actually **a module** of *975 lines of Python code*.
- This module provides a standard interface to extract, format and print stack traces of Python programs.

Introduction (review)

- It exactly mimics the behavior of the Python interpreter when it prints a stack trace.
- In this way, it allows you to follow the line and character of the error and trace it.

Tips:

- Concentrate on the **last lines** of the error messages.

Introduction (review)

- The name of module - ***Traceback*** - appears when your code causes an error and it reports detailed information on that specific error, demonstrating the particular files in which the error occurred.

```
Traceback (most recent call last):
File "code.py", line 1
    print("Don't say 'I never make a mistake'"
                                               ^
SyntaxError: unexpected EOF while parsing
```

- In these error messages, the most important thing that a programmer should be interested in is **the last lines** in the most cases.

Introduction (review)

- In this example, the last two lines indicate that this error type is a Syntax error and it also indicates in which line and in which character (with the ^ sign) the error raised.

```
1  Traceback (most recent call last):
2  File "code.py", line 1
3      print("Don't say 'I never make a mistake!"
4                                                ^
5  SyntaxError: unexpected EOF while parsing
```

⚠ Attention:

- Do not panic when you see those error lines. Do not hesitate to read carefully what they are saying to you.

2

Syntax Errors

Syntax Errors (review)

- In the previous example (shown below), you must have seen the mysterious word **SyntaxError**, which you will likely encounter frequently during your time in Python.

```
Traceback (most recent call last):
File "code.py", line 1
    print("Don't say 'I never make a mistake'"
                                               ^
SyntaxError: unexpected EOF while parsing
```

- A **wide variety of errors** in Python are called `SyntaxError`. Typically, they indicate a problem that Python encountered when trying to compile your program, or that your code could not be run.

Syntax Errors (review)

- Every syntax error has a text value (**known as associated value**) that describes the error in detail.
- In this example, the message "**SyntaxError: unexpected EOF while parsing**" means that something else had been expected by the interpreter after your statement, but you didn't pass it to the interpreter.

```
1  Traceback (most recent call last):
2  File "code.py", line 1
3      print("Don't say 'I never make a mistake'"
4                                                ^
5  SyntaxError: unexpected EOF while parsing
```

) is forgotten

associated value

3 Common Errors

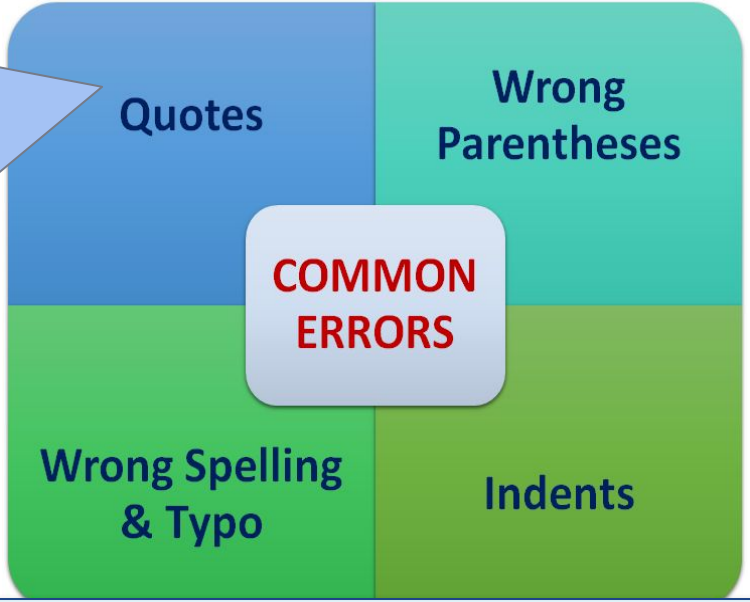
Summarize what you've learned from pre-class content about common errors:

Students, write your response!

Common Errors (review)

Common Errors (review)

In the **Matter of Quotes** lesson, we have elaborated on how sensitive programming language Python is to **quotation** marks. So it's critical not to forget to enclose a string in quotes of the same type.



Tips:

- Keep this simple advice in your mind; triple quotes for multi-line strings, double or single quotes for ordinary strings.

Common Errors (review)

One of the most common syntax errors is the wrong number of parentheses in function calls. e.g.

```
print(len(my_list[2:])
```

Common Errors (review)

Yes, it may sound strange to you, but the **most common mistake** made by programmers is the wrong spelling keywords, function names, and variable names. e.g. `True` and `true`, `print` and `prit` or `pirnt`.

⚠ **Avoid ! :**

- Do not confuse uppercase and lowercase letter of the keywords. Keep in your mind that Python is a case-sensitive programing language.

Common Errors (review)

Indents are also very common errors for programmers.

⚠ **Avoid ! :**

- Do not forget to put the appropriate indent where necessary. Keep in your mind that Python is a indent-sensitive programing language.

Common Errors

- Let's find some errors in the codes :

```
1  status = []
2  if status:
3      print("'''Hello World")
4  else
5      print("Hello Universe'")
6
```

What is the error? Try to figure out in your mind...

Students, write your response!

Common Errors

- There is a typo (missing colon) :

```
1  status = []
2  if status:
3      print("'Hello World")
4  else
5      print("Hello Universe'")
6
```

a colon **:** should be put here

Output

```
  File "code.py", line 4
    else
       ^
SyntaxError: invalid syntax
```

Common Errors

- Let's find some errors in the codes :

```
1  x = ["1", "2", "3"]
2  y = ["USA", "Japan", "Spain"]
3
4  for i in y:
5      for j in X:
6          print(type([tuple(i+j)]))
7
```

What is the error? Try to figure out in your mind...

Students, write your response!

Common Errors

- Remember, Python is a case-sensitive language :

```
1  x = ["1", "2", "3"]
2  y = ["USA", "Japan", "
3
4  for i in y:
5      for j in X:
6          print(type([tuple(i+j)]))
7
```

case of "x" should be the same

Output

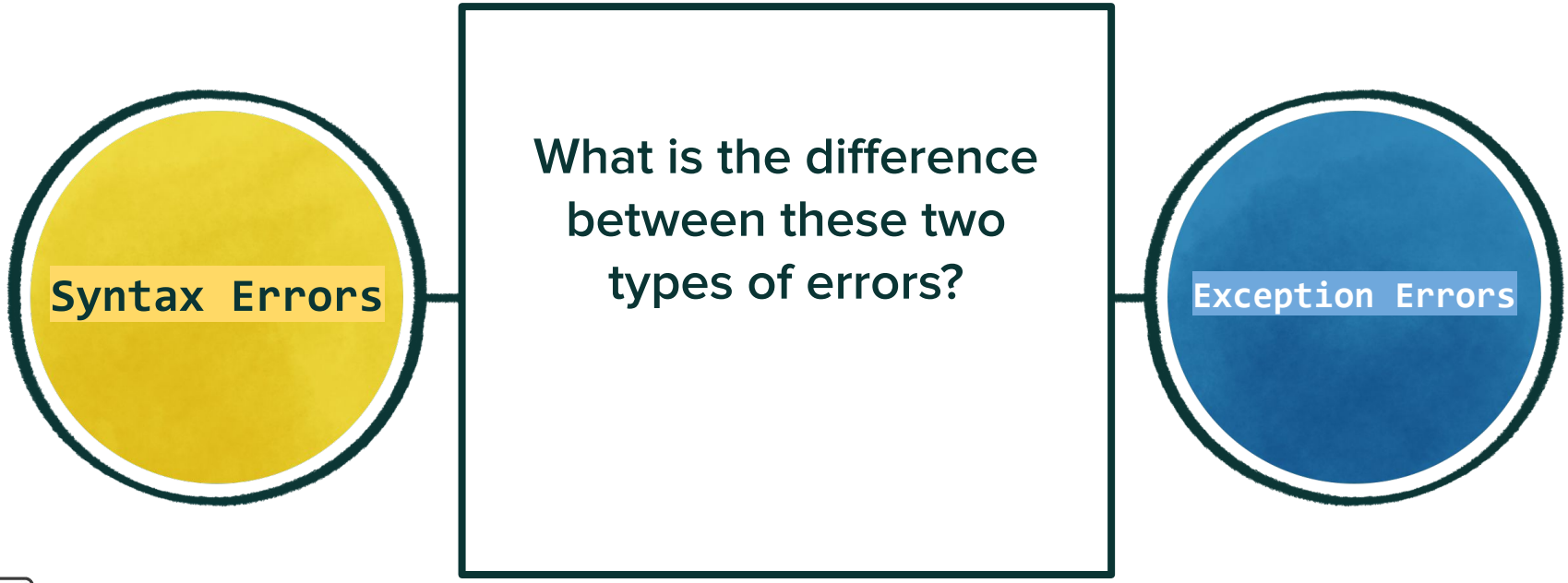
```
Traceback (most recent call last):
  File "code.py", line 5, in <module>
    for j in X:
NameError: name 'X' is not defined
```

Exceptions

Table of Contents

1 Introduction

What is the difference?



Introduction (review)

- If your program is running but you still get some error messages for several reasons, you probably get an error warning called an **`exception`**. This is because of defective line of codes in your program.
- What is the difference between `Syntax errors` and `Exception errors`?

Syntax Error	Exception Error
These types of errors are detected during **compiling** the program into byte-code.	These types of errors are detected during the program **execution** (**interpretation**) process.

Introduction (review)

- Now, let's examine the following example :

```
1  print('Here we go!')
2  print('I will be the second text')
3  a = '3'
4  b = 5
5  print('It is time for an error message :(')
6  print(a + b)  # it won't be printed
7  print("Sorry, but I won't be printed")  # it won't ve printed
```

Introduction (review)

- Now, let's examine the following example :

```
1  print('Here we go!')
2  print('I will be the second text')
3  a = '3'
4  b = 5
5  print('It is time for an error message :(')
6  print(a + b)  # it won't be printed
7  print("Sorry, but I won't be printed")  # it won't ve printed
```

```
1  Here we go!
2  I will be the second text
3  It is time for an error message :(
4  Traceback (most recent call last):
5    File "code.py", line 6, in <module>
6      print(a + b)
7  TypeError: can only concatenate str (not "int") to str
```

Introduction (review)

- The first three `print()` functions printed their own content, but the last two `print()` functions did not work and we received an `exception error` message (that is our program was partially executed).

```
1  Here we go!
2  I will be the second text
3  It is time for an error message :(
4  Traceback (most recent call last):
5    File "code.py", line 6, in <module>
6      print(a + b)
7  TypeError: can only concatenate str (not "int") to str
```

- As we stated before, you should look at the last line of the error message which says that this is the TypeError. It's obvious, right? There is a wrong use of types of variables (`a` is a `str` type and `b` is an `int`).

Introduction (review)

- Similar to the syntax errors, almost all `exception error` messages also have a text value (**associated value**) following the error message explaining what the error is related to.

```
1  Here we go!
2  I will be the second text
3  It is time for an error message :(
4  Traceback (most recent call last):
5    File "code.py", line 6, in <module>
6      print(a + b)
7  TypeError: can only concatenate str (not "int") to str
```

Explanatory text value of the error message. It's known as **"associated value"**

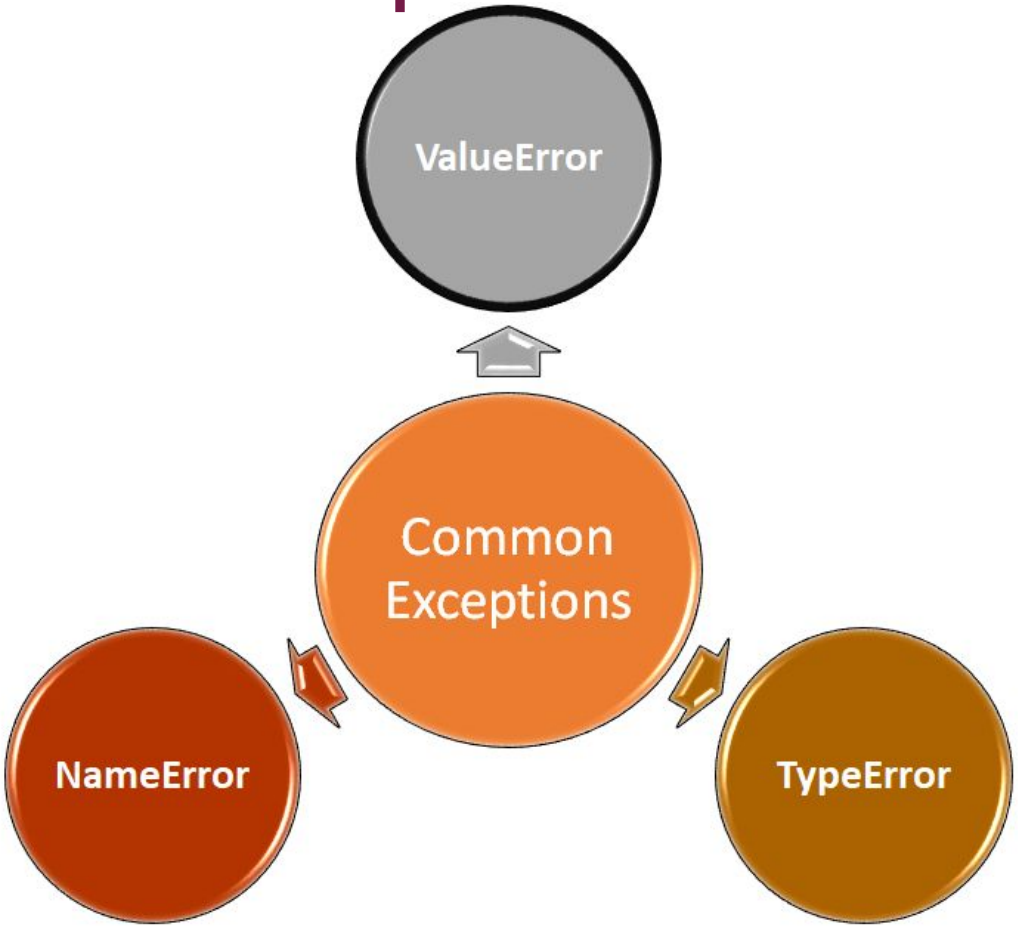
2

Common Exceptions

Can you summarize the common exceptions?

Students, write your response!

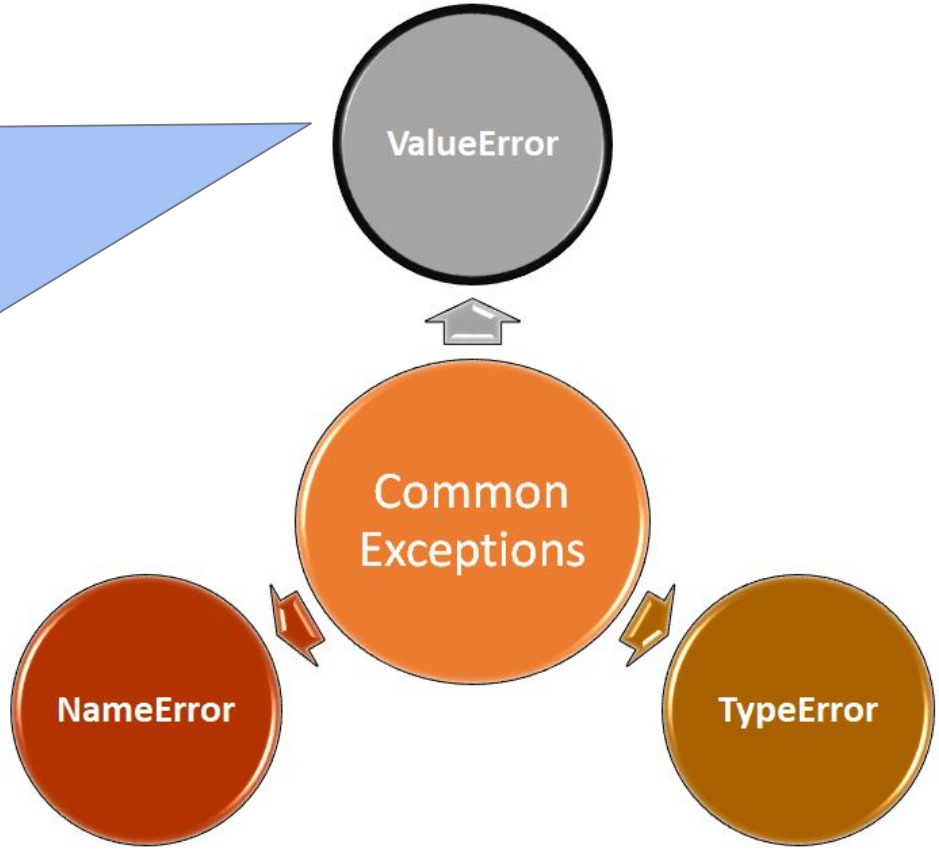
Common Exceptions (review)



Common Exceptions (review)

-Raised when an operation or function receives an argument that has the right type but an inappropriate value, and the situation is not described by a more precise exception such as `IndexError`.

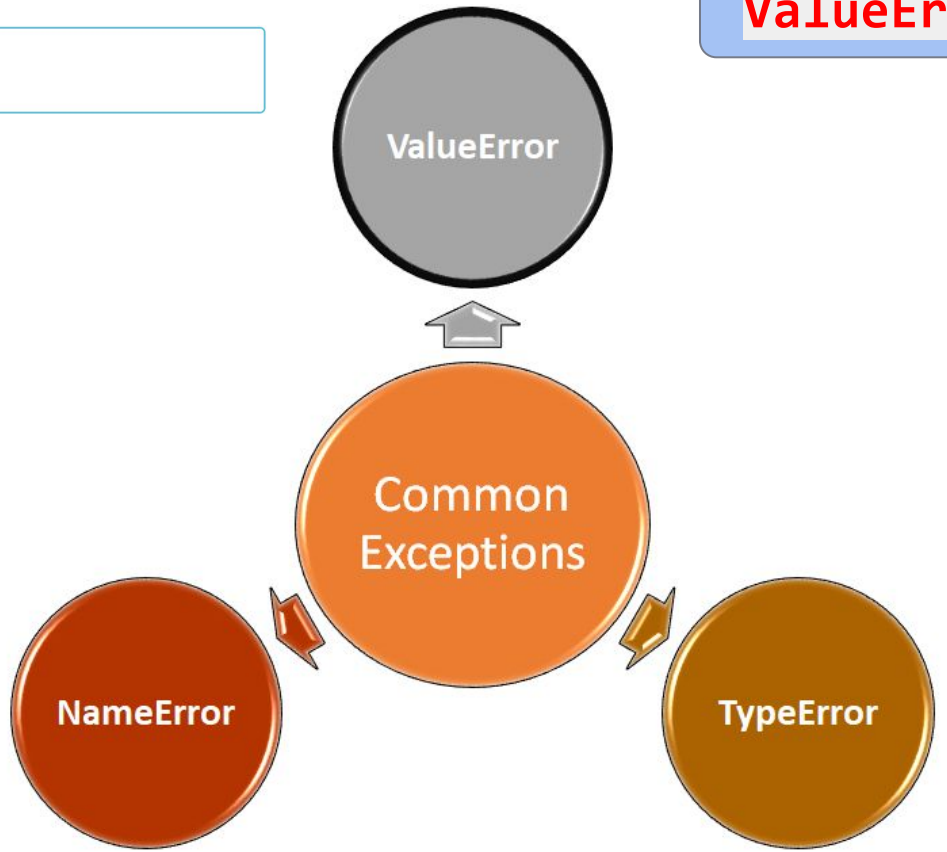
-In other words; to encounter a `ValueError` in Python means that it is a problem with the content of the object you tried to assign the value to.



Common Exceptions (review)

```
1  print(int('ten'))
2
```

ValueError

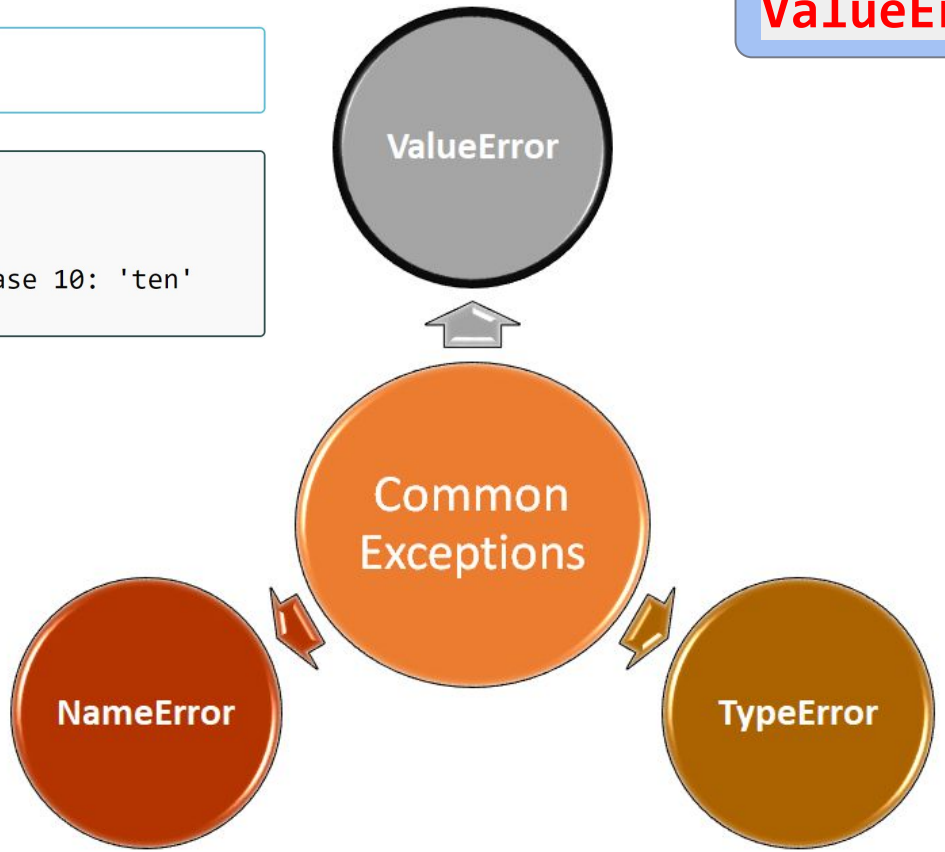


Common Exceptions (review)

```
1  print(int('ten'))
2
```

```
1  Traceback (most recent call last):
2    File "code.py", line 1, in <module>
3      print(int('ten'))
4  ValueError: invalid literal for int() with base 10: 'ten'
5
```

ValueError



Common Exceptions

- **Task :**
  - Try to set a code to raise `ValueError` intentionally using the `math` module.

Common Exceptions

- A **sample** of the code can be as follows :

ValueError

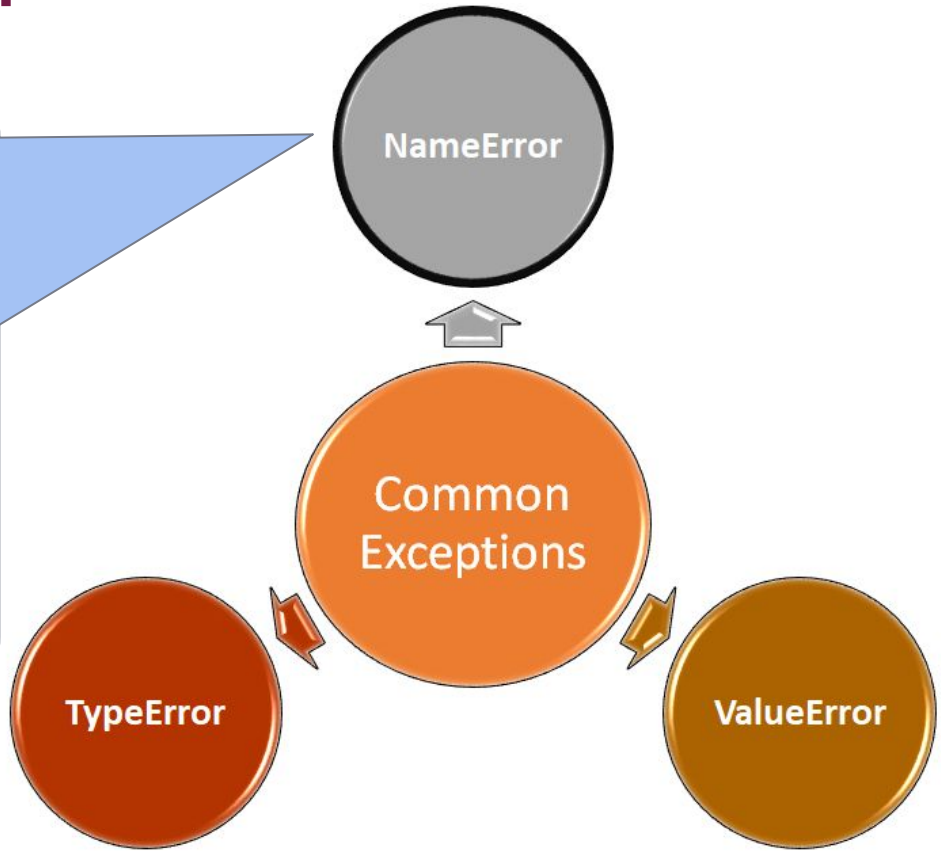
```
1  import math
2  print("I am trying to get the square root of a negative number.")
3  print(math.sqrt(-10))
4
```

Output

```
I am trying to get the square root of a negative number.
Traceback (most recent call last):
  File "code.py", line 3, in <module>
    print(math.sqrt(-10))
ValueError: math domain error
```

Common Exceptions (review)

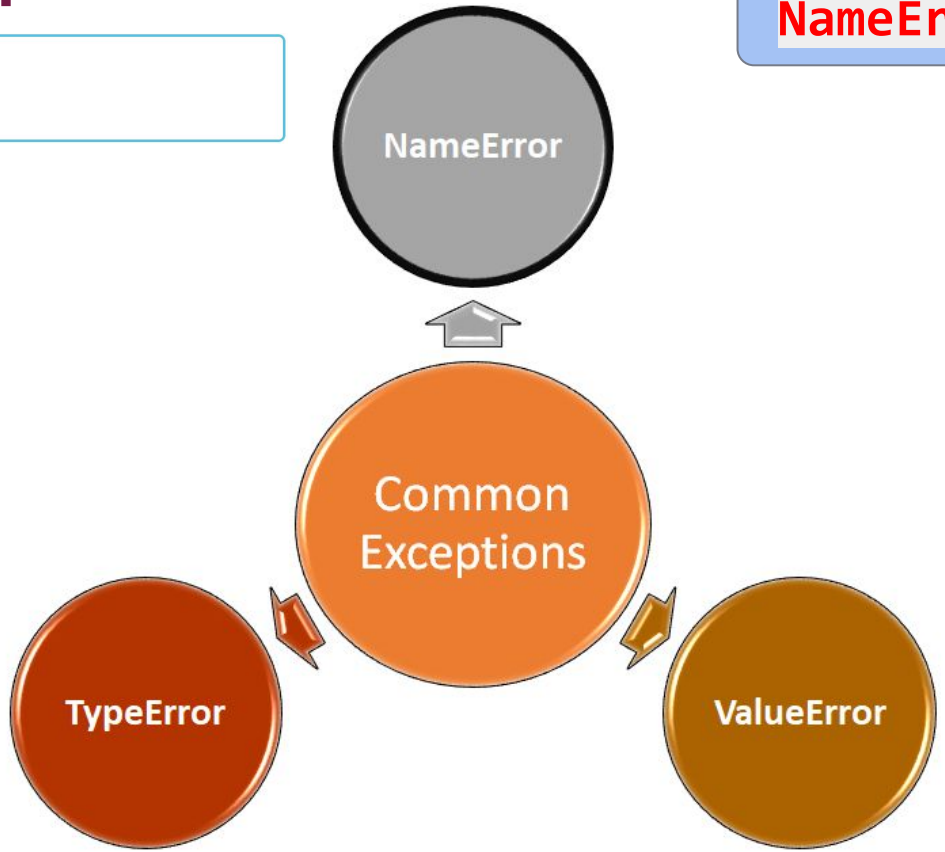
-This error is usually raised if a variable you use in the code stream is not pre-defined or not properly defined.
-In other words, it is raised when a local or global name is not found. This applies only to unqualified names.
-The associated value is an error message that includes the name that could not be found.



Common Exceptions (review)

```
1  print(variable)
2  variable = "Don't ever give up!"
```

NameError

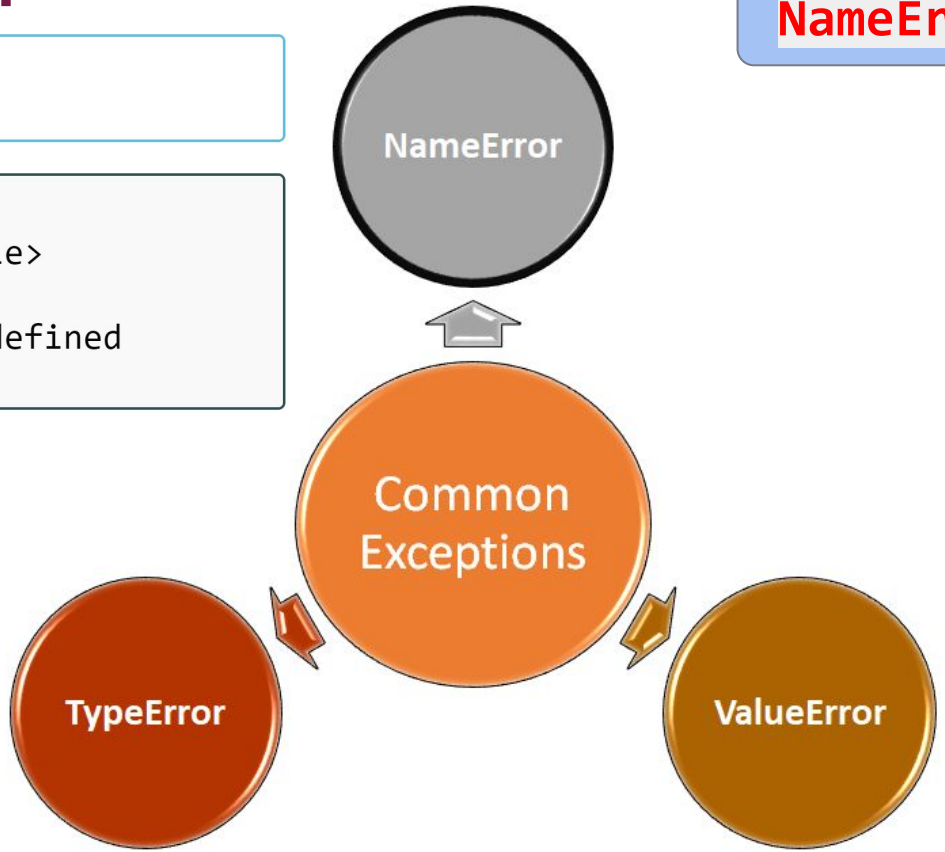


Common Exceptions (review)

```
1  print(variable)
2  variable = "Don't ever give up!"
```

```
1  Traceback (most recent call last):
2    File "code.py", line 1, in <module>
3      print(variable)
4  NameError: name 'variable' is not defined
5
```

NameError

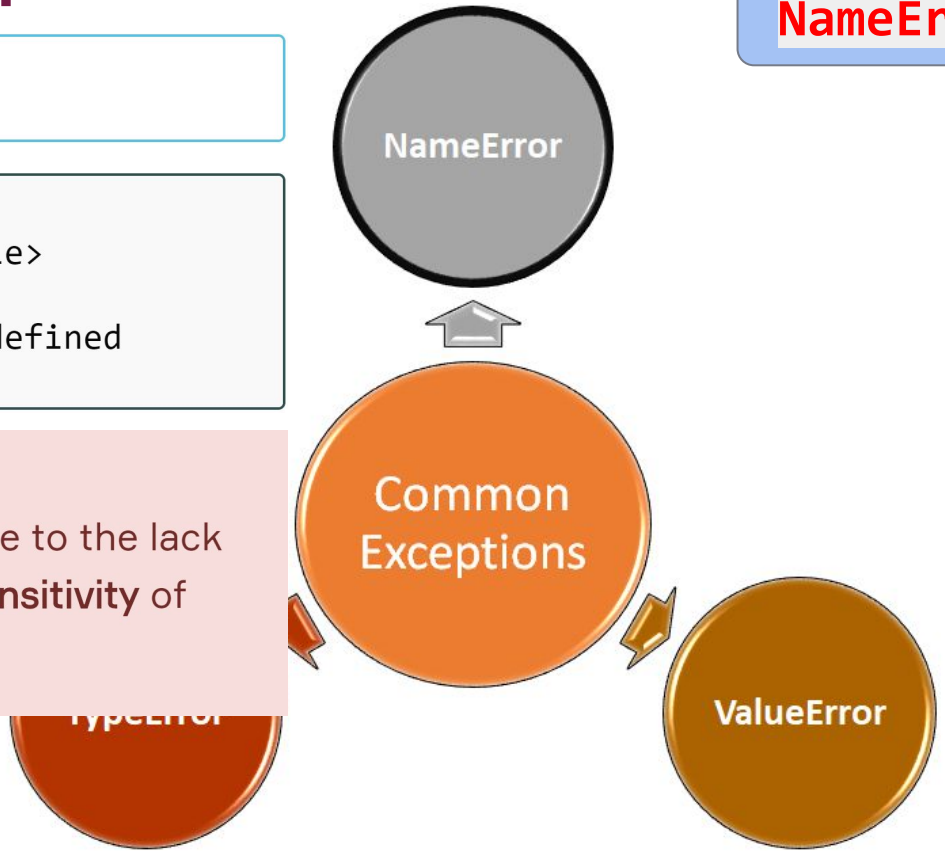


Common Exceptions (review)

NameError

```
1  print(variable)
2  variable = "Don't ever give up!"
```

```
1  Traceback (most recent call last):
2    File "code.py", line 1, in <module>
3      print(variable)
4  NameError: name 'variable' is not defined
5
```



⚠ **Attention :**

- Note that the NameError often raises due to the lack of attention to these two things: **case-sensitivity** of Python and **pre-defines** of the variables.

Common Exceptions

- **Task :**
  - Try to set a code to raise `NameError` intentionally.

Common Exceptions

NameError

- A **sample** of the code can be as follows :

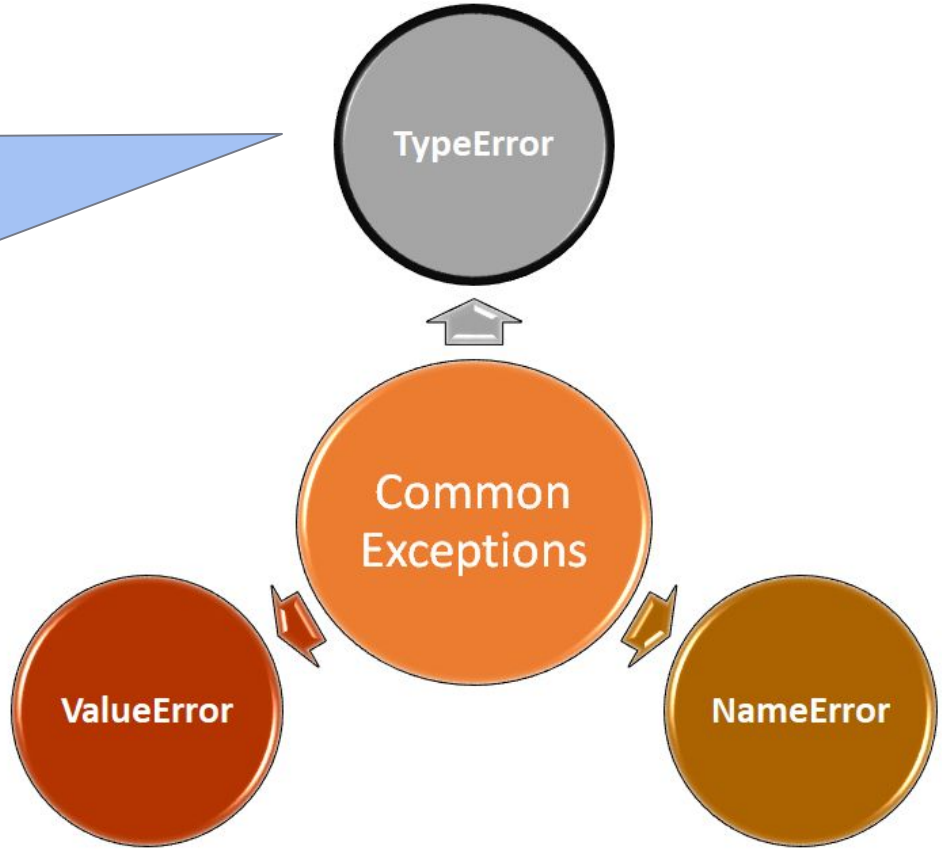
```
1  b = "string value 1"
2  c = "string value 2"
3  print(b)
4  print(C)
5
```

Output

```
string value 1
Traceback (most recent call last):
  File "code.py", line 4, in <module>
    print(C)
NameError: name 'C' is not defined
```

Common Exceptions (review)

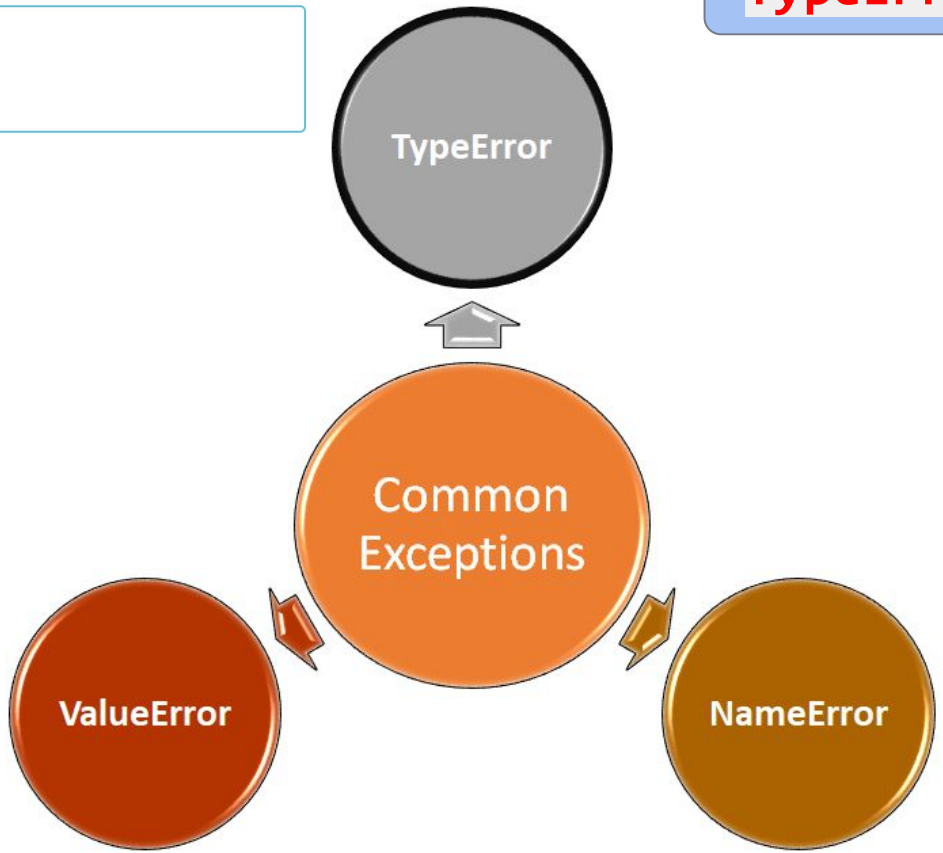
-Raised when an operation or function is applied to an object of inappropriate type.
-The associated value is a string giving details about the type mismatch.



Common Exceptions (review)

```
1 for i in range('x'):
2     print(i)
3
```

TypeError

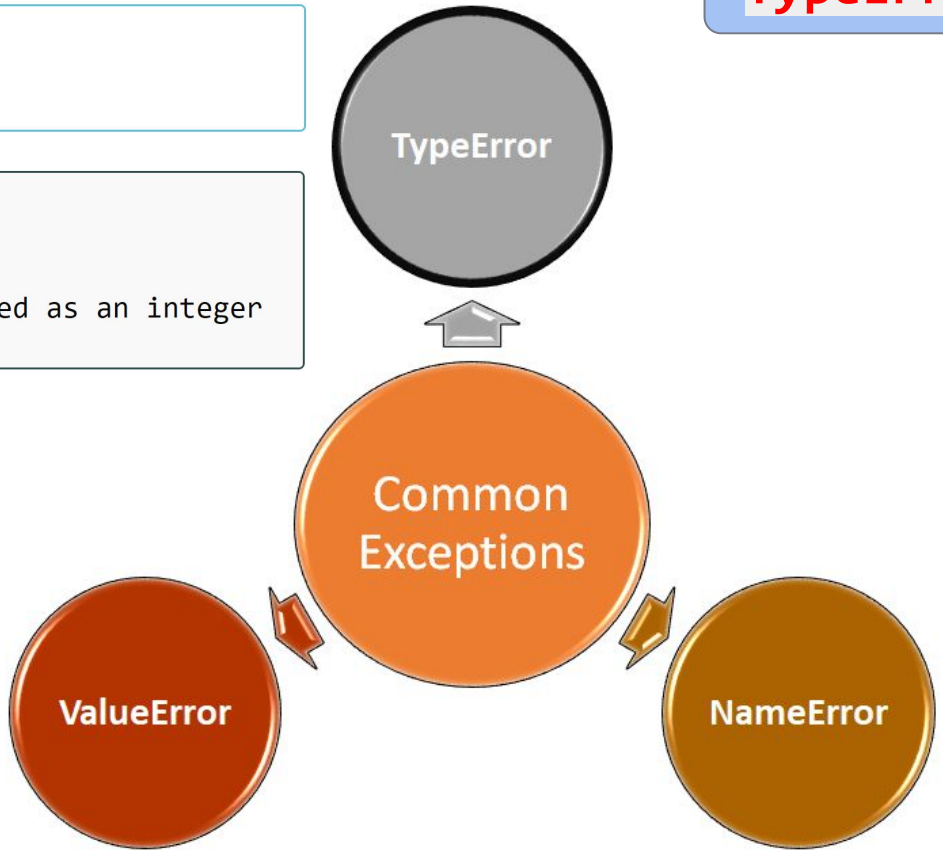


Common Exceptions (review)

```
1  for i in range('x'):
2      print(i)
3
```

```
1  Traceback (most recent call last):
2    File "code.py", line 1, in <module>
3      for i in range('x'):
4  TypeError: 'str' object cannot be interpreted as an integer
5
```

TypeError



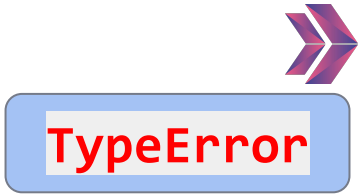
Common Exceptions (review)

- As in the previous example; although it requires `int`, we have tried to iterate the wrong type - `str` - object in the `range()` function.

Tips:

- The most useful way you can do about all the errors you can't deal with yourself is to search for the error message on the internet search engine.
- You can make sure that the errors that you will encounter and their solutions have been experienced by someone previously.

Common Exceptions



- **Task :**
  - Try to set a code to raise `TypeError` intentionally.

Common Exceptions

- A **sample** of the code can be as follows :

TypeError

```
1 print(2 + "2")
2
```

Output

```
Traceback (most recent call last):
  File "code.py", line 1, in <module>
    print(2 + "2")
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

THANKS!

Any questions?