LECTURER: TAI LE QUY

# PROGRAMMING WITH PYTHON

## TOPIC OUTLINE

TOPIC OUTLINE

## Version Control

6

UNIT 3

## ERRORS AND EXCEPTIONS

## STUDY GOALS

- Understand how to define program errors and get the exception information
- Identify errors in Python programs and understand how to deal with them in a principled manner
- Determine how to handle and raise exceptions
- Learn how to create custom user-defined exception classes

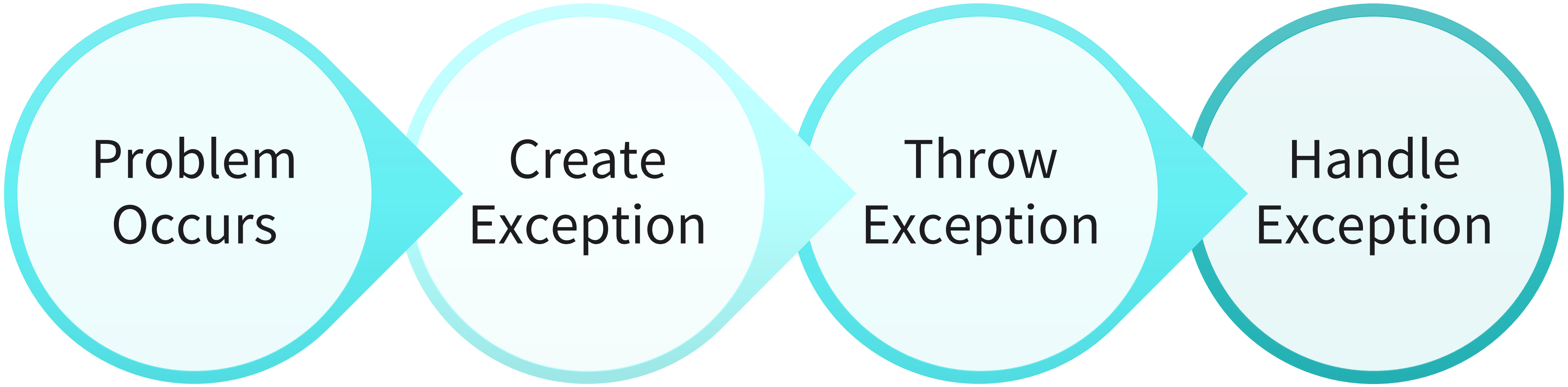
EXPLAIN SIMPLY

1. Describe the three types of errors that most commonly happen when programming?

2. How does Python handle exceptions?

3. How can programmers create user-defined exceptions?

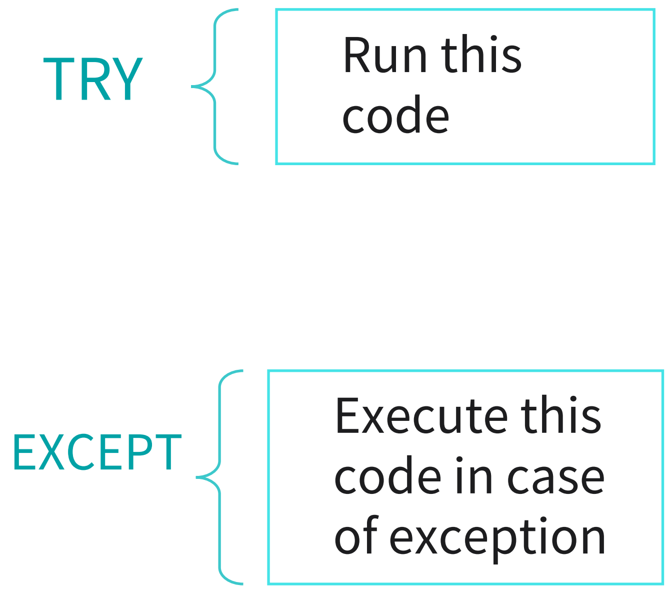
- **Objective:** write programs where everything goes as expected
- **Reality:** problems occur



- **Approach:** use exceptions to deal with errors

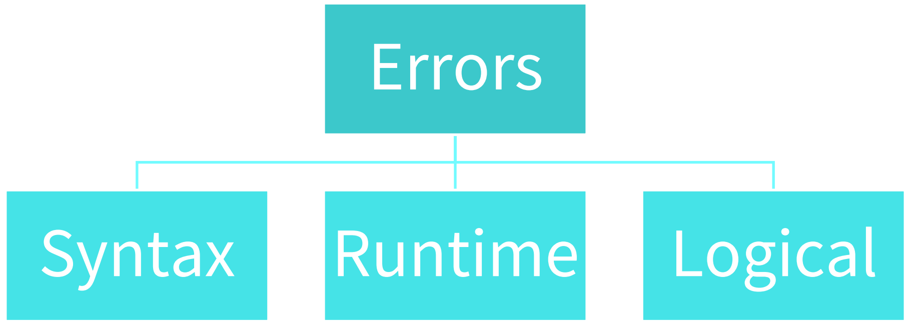
## **Exception handling:** respond to errors occurring during execution

- Handle errors using built-in or custom-created exceptions
- Rely on Python’s well-defined code block organization



## Three types of errors occur when programming

- **Syntax:** not conform to rules of Python programming language
- **Runtime:** errors during program execution
- **Logical:** not following implementation requirements



## Examples of three types of errors

**Syntax**

```
def main():
    print(1/1))

if __name__ = '__main__':
   main()
```

Invalid syntax

**Runtime**

```
def main():
    print(1/0)

if __name__ = '__main__':
   main()
```

Division by zero

**Logical**

```
def main():
    a = "Python"
    if a == "Python"
        print("a is equal
         to Java")

if __name__ = '__main__':
    main()
```

Logical flaw

## Function sys.exc_info() provides additional information

exception_type, exception_value, exception_traceback = sys.exc_info()

### Type

- Type of the exception  being handled

### Value

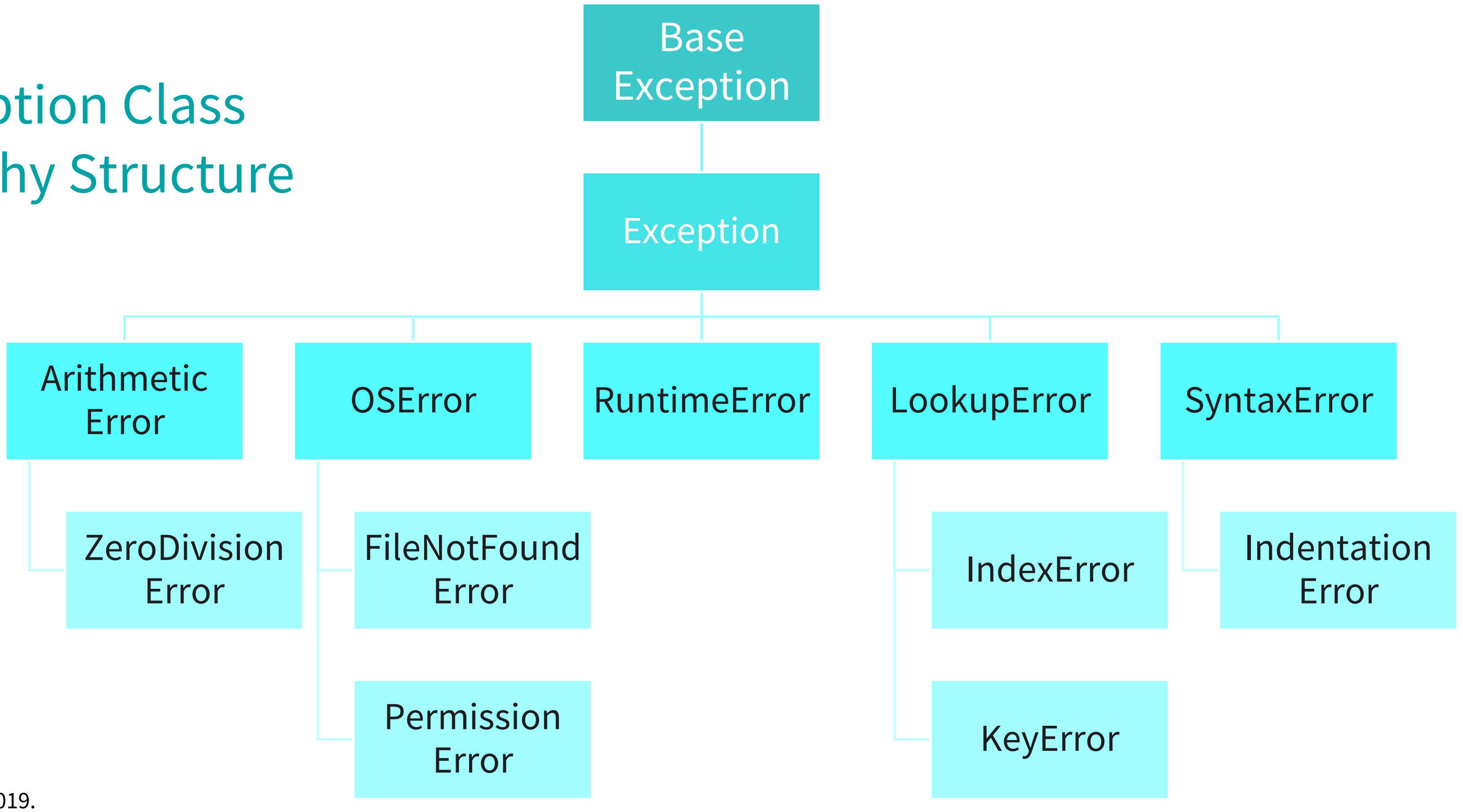
- Exception instance type

### Traceback

- The call stack at the point where the exception occurred

## Custom-created exception classes follow class hierarchy structure

Exception Class Hierarchy Structure



Source of the text: Mathes, 2019.

## Handle exceptions using **try – except – finally** block

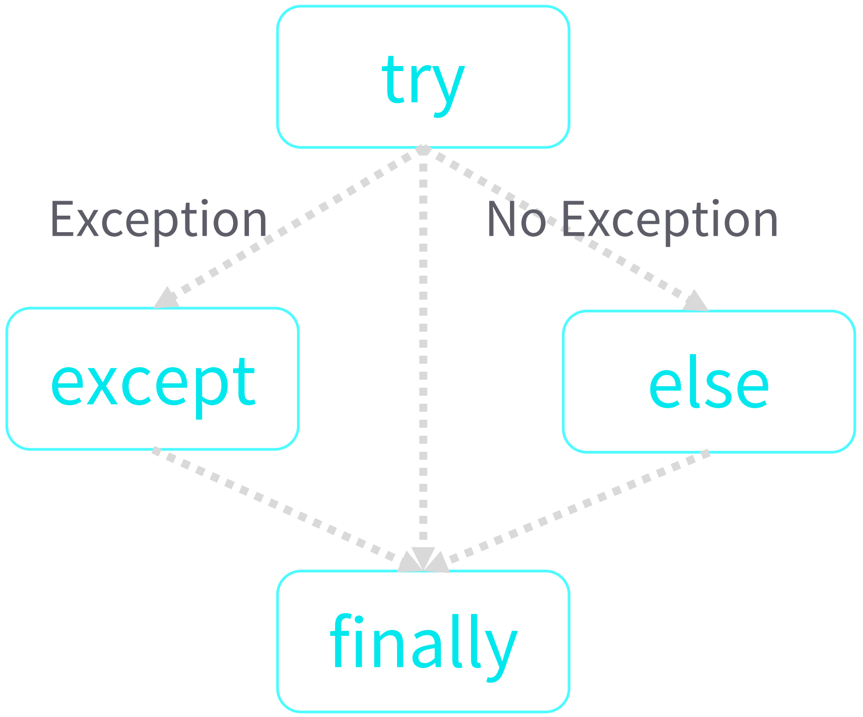
- **try:** code that gets executed
- **except:** program’s response to exceptions
- **finally:** occurs regardless if an exception is thrown or not

```
def handle_exceptions():
    try:
      # function code
    except:
     # handle exceptions
    finally:
     # code clean - up
```

Source of the text: Lutz, 2017.

## THE ELSE CLAUSE

**else clause:** enables execution of code block when exception does not occur



```
def division_by_zero():
    try:
        division = 1 / 1
    except:
        info = get_exception_info()
        print(info)
    else:
        print(“No exception occurred”)
    finally:
        pass
```

## RAISING AN EXCEPTION

**raise statement:** used to create custom exception messages

- Diverts execution in a matching except suite, or stops the program when no matching except suite is found
- Implemented for data control and validation

```
def main():
    a = “Java”
     if a != “Python”:
          raise Exception(“a is not
                    equal to Python”)

if __name__ == ‘__main__’:
    main():
```

## Inheriting from exceptions allows to create custom exceptions

- User-defined exceptions require careful design and lots of effort
- Note: custom exceptions are rarely used and prone to errors

```
class MyException(Exception):
    def __init__(self, *args, **kwargs):
        super().__init__.(self, *args, **kwargs)

class MyIndexError(IndexError):
    def __init__(self, *args, **kwargs):
        super().__init__.(self, *args, **kwargs)
```

## REVIEW STUDY GOALS

- Understand how to define program errors and get the exception information
- Identify errors in Python programs and understand how to deal with them in a principled manner
- Determine how to handle and raise exceptions
- Learn how to create custom user-defined exception classes

UNIT 3

## TRANSFER TASK

## **Expect the unexpected:** handle exceptions in Python

Consider a simple example that asks the user to enter the name of a student to display her/his age.

**Modify** the provided code to validate the provided input by using exceptions. How does this improve code efficiency?

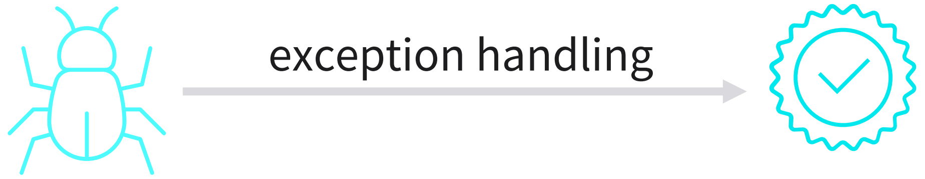
```
students = {“Emily’: 22, David: “19”}

def print_student_age():
    name = input(“Enter name of student”)
    print(students[name])

print_student_age()
```

## TRANSFER TASK: SAMPLE SOLUTION

- **try – except – finally** blocks allow to handle exceptions
- Dealing with errors in a principled manner improves the robustness of our programs



```
students = {"Emily": 22, "David": 19}
def print_student_age():
  while True:
     try:
        name = input("Enter name of student: ")
        print(students[name])
        break
     except:
        print("Name not registered")
     finally:
        print("...terminating")
print_student_age()
```

## TRANSFER TASK
## PRESENTATION OF RESULTS

Please present your results.
The results will be discussed in plenary.

LEARNING CONTROL QUESTIONS

1. Which of the following is a collection of errors defined in Python?
   a) syntax, runtime, logic errors
   b) systematic, runtime, conditional errors
   c) script, runtime, conditional errors
   d) runtime, unconditional, conditional errors

2. Exception is defined in Python as …
   a) ... an error that happens during the preprocessing of a program
   b) ... an error that happens during the interpretation of a program
   c) ... an error that happens during the compilation of a program
   d) ... an error that happens during the execution of a program

3. In Python, the exceptions are handled using …
   a) …try – except – end blocks
   b) …try – error – finally blocks
   c) …try – except – finally blocks
   d) …try – else – finally blocks

## LIST OF SOURCES

Mathes, E. (2019). *Python crash course* (2nd ed.). No Starch Press.
Lutz, M. (2017). *Learning python* (5th ed.). O’Reilly.