LECTURER: TAI LE QUY

# PROGRAMMING WITH PYTHON

# Who am I?

- Name: Tai Le Quy
- PhD candidate at L3S Research Center – Leibniz University Hannover
  - Topic: Fairness-aware machine learning in educational data mining
  - Project: LernMINT (lernmint.org)
- MSc in Information Technology at National University of Vietnam
- Profile: tailequy.github.io
- Email: tai.le-quy@iu.org
- Additional materials: https://github.com/tailequy/IU-PythonProgramming

# Who are you?

- Name
- Employer
- Position/responsibilities
- Fun Fact
- Previous knowledge? Expectations?

TOPIC OUTLINE

TOPIC OUTLINE

# Version Control

6

UNIT 1

# INTRODUCTION TO PYTHON

**STUDY GOALS**

- Learn fundamental building blocks of Python programming language
- Describe different data structures and their specific properties
- Understand how to design and implement functions
- Distinguish different types of data input and output
- Identify how to control and change data flow
- Know how to create modules and packages

**EXPLAIN SIMPLY**

1. What are examples of primitive and non-primitive data structures in Python?

2. What is the purpose of functions? What types of functions exist in Python?

3. How can one control the flow of the program execution through “if” statements and loops?

## WHY LEARN THE PYTHON PROGRAMMING LANGUAGE?

### MOST POPULAR

Extremely versatile with multiple uses

Fast growing

### EASY AND POWERFUL

Easy to code, maintain and learn

General-purpose language

**WHY**
Python?

### EFFICIENT

Used in a variety of scenarios and tasks

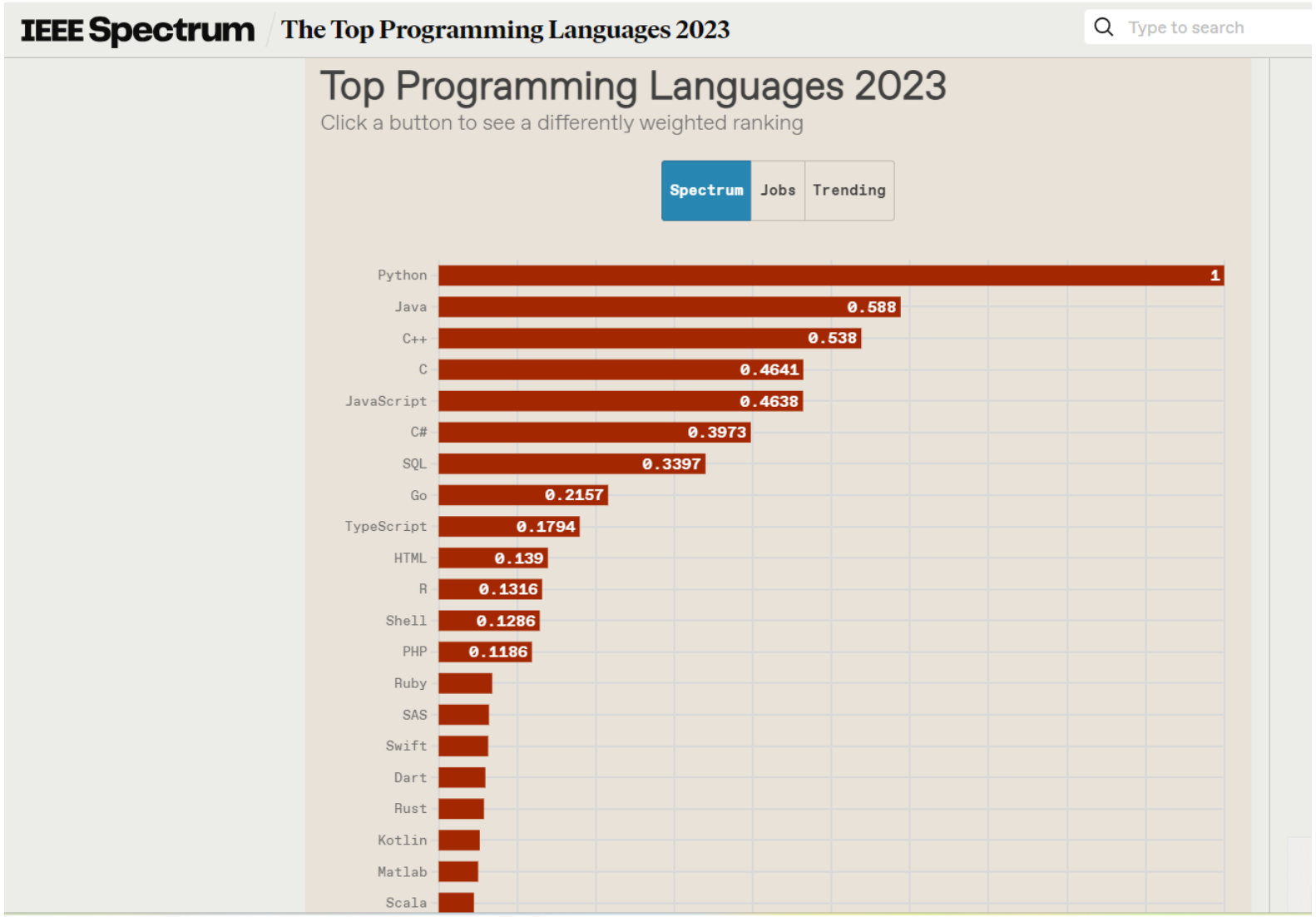
Large supportive community

### ROBUST

Robust core standard library

Many open source frameworks

## THE TOP PROGRAMMING LANGUAGES




Source: https://spectrum.ieee.org/the-top-programming-languages-2023#toggle-gdpr

# **Primitive variables:** most basic data structures

### INTEGERS
Numerical data including whole numbers (Example: year = 2022)

### FLOATS
Floating point numbers for rational numbers with a decimal point (Example: pi = 3.14)

### STRINGS
Sequence of characters denoted using single or double quotes (Example: sentence = "Hello World!")

### BOOLEAN
Two logical values: True and False (Example: is_python_cool = True)

# Non primitive data structures:
ordered and unordered sequences of objects

## LISTS
Ordered sequences of objects enclosed in brackets. Example: list_example = [1, 2, “a”, “b”]

## TUPLES
Ordered sequences of objects enclosed in parantheses. Example: tuple_example = (1, 2, “a”, “b”)

## DICTIONARIES
Ordered sequences of objects that are key-value paired, enclosed in curly brackets.
Example: dictionary_example = {1: “a”, 2: “b”})

## SETS
Unordered sequences of unique objects. Example: set_example = set([1, 2, 3, 4, 5])

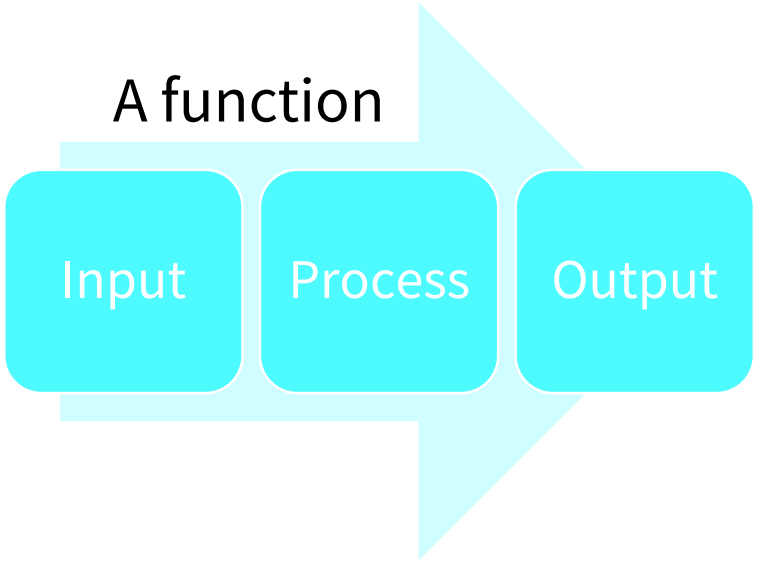
## FROZEN SET
Immutable set objects. Example: frozenset_example = frozenset([1, 2, 3, 4, 5])

**STRING MANIPULATION**

Strings = series of characters.
Example: sentence = “I love Python”

| | |
|---|---|
| Method len() calculates number of characters | print(**len**(sentence)) # 13 |
| String concatenation using the + or join() | sentence.join(“so much”)<br># “I love Python so much” |
| count() counts how many times a string appears in another | sentence.count(“o”) # 2 |
| lower() / upper() convert a string to all lowercase / uppercase | sentence.upper()<br># I LOVE PYTHON |

A **function:** reusable block of code used to solve a predefined task




Example

```
def function_name (parameters):
        """
        function comments
        """
        <statement(s)>

        return <expression>
```

**TYPES**

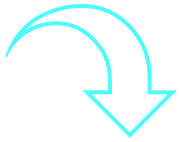
- **BUILT-IN**
  Developed inside Python interpreter
- **USER-DEFINED**
  Developed by user
- **ANONYMOUS**
  Defined using “lambda” keyword

Functions encapsulate computations to be treated as primitives

- Function parameters provided as input.
  **Default** parameters assigned using the = operator
- **Variable** number of parameters denoted by
  single or double *
- **Lambda** functions: quick and efficient forms of functions

Source of the text: Mathes, 2019.

Example: <payment> has parameters <hours> and <wage> with default value of None



```
def payment(hours, wage=None):
  if wage is not None:
    total = hours * wage
  else:
    total = 0
  return total
```

Variable number of parameters

```
def sum_number(*args):
  total_sum = sum(args)
  return total_sum
```

```
def main():

  f_double = lambda x: x * 2
  print(f_double(10))
```

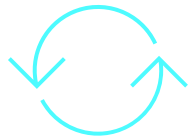
Lambda function

**Conditional statements:** perform computations based on evaluation of a Boolean expression

- Use Boolean expression to control code flow
- If <expression> is True, then <statement> is executed. Otherwise <statement> is skipped
- Else and else-if (elif) conditions can be combined with “if” statement

**Repetition statements (loops):** execute lines of code multiple times depending on specific conditions

- “for” loop to iterate over any data structure
- “while” loop repeats as long as a Boolean condition is true
- “break” statement to stop the loop before it ends
- “continue” statement skips the current block and continues the loop

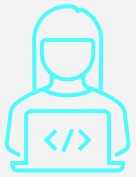


Source of the text: Fabrizio, 2018

## For Loop Example:

**break** and **continue** used to decide which elements to display

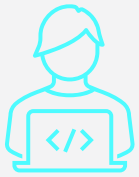
```
def main():
  ebooks = ["python","perl","ruby"]
  for item in ebooks:
    if item = "ruby":
            break
    else:
            print(item)
            continue
```



## While Loop Example:

elements displayed when matching certain conditions

```
def main():
  ebooks = ["python","perl","ruby"]
  i = 0
  while ebooks[i] != "ruby"
            print(ebooks[i])
            i += 1
```

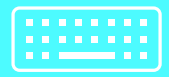


Note: <break> and <continue> are rarely used

## Functions

- input() to interact with the user;
  printf() to output information entered by the user

## FILE IO



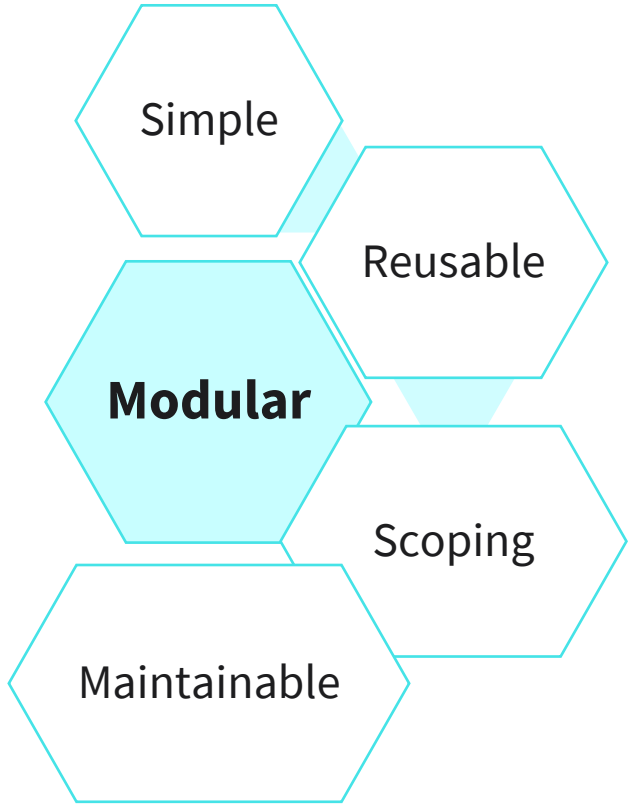
- open(), read, write() and close() used for opening, reading from, writing into and closing a file

## PATHS OF FILES

- pathlib library to address location paths

A **module:** a Python file that defines functions, classes and variables

- Modular programming: divide large programs into smaller modules
- Use individual modules as building blocks for larger applications



## REVIEW STUDY GOALS

- Learn fundamental building blocks of Python programming language
- Describe different data structures and their specific properties
- Understand how to design and implement functions
- Distinguish different types of data input and output
- Identify how to control and change data flow
- Know how to create modules and packages

UNIT 1

# TRANSFER TASK

TRANSFER TASK

# Group Work: Built-In Data Structures in Python

**READ**

Organize the information you have about built-in data structures in Python such as List, Tuple, Dictionary and Sets

**IDENTIFY**

Identify key properties and functionalities of each of the above built-in data structures

**EXPLAIN**

Describe (conceptually) in which kind of scenarios you would use a particular built-in data structure

**DISCUSS**

Discuss your findings and compare them with the other groups

**TRANSFER TASK**
**PRESENTATION OF RESULTS**

Please present your results.

The results will be discussed in plenary.

1. What kind of language is Python
    a) Typed language
    b) Static language
    c) Interpreter language
    d) Compiler language

2. Which of the following is a collection of primitive data structures in Python
   a) integers, floats, strings, boolean
   b) doubles, floats, strings, boolean
   c) integers, floats, strings, decimals
   d) integers, decimals, strings, boolean

3. Which of the following is a collection of non-primitive data structures in Python
   a) arrays, tuples, dictionaries, sets, frozensets
   b) lists, tuples, dictionaries, sets, frozensets
   c) lists, arrays, dictionaries, sets, tuples
   d) lists, dictionaries, tables, sets, frozensets

## LIST OF SOURCES

Mathes, E. (2019). *Python crash course* (2nd ed.). No Starch Press.
Fabrizio, R. (2018). *Learn python programming* (2nd ed.). Packt Publishing.