

**LECTURER: TAI LE QUY**

# **ALGORITHMS, DATA STRUCTURES, AND PROGRAMMING LANGUAGES**

## Who am I?

- Name: Tai Le Quy
- PhD at L3S Research Center – Leibniz University Hannover
- Topic: Fairness-aware machine learning in educational data mining
- MSc in Information Technology at National University of Vietnam
- Profile: [tailequy.github.io](https://github.com/tailequy)
- Email: [tai.le-quy@iu.org](mailto:tai.le-quy@iu.org)
- Materials: <https://github.com/tailequy/IU-Algorithm>



## INTRODUCTORY ROUND

# Who are you?

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



TOPIC OUTLINE

Basic Concepts

1

Data Structures

2

Algorithm Design

3

Basic Algorithms

4

Measuring Programs

5

---

**Programming Languages**

---

**6**

**Overview of Important Programming Languages**

**7**

**UNIT 1**

# **BASIC CONCEPTS**



- Explain the role of algorithms, data structures and programming languages in programming.
- Represent algorithms in various ways.
- Understand the role of abstraction and encapsulation in programming.
- Define concepts of control structures in programming languages.
- Differentiate between different data types.
- Create basic data structures: List, Chain, Tree.



1. Describe briefly different ways in which algorithms are specified.
2. Explain what you understand by data encapsulation.
3. State the attributes based on which a good language is chosen.



- Algorithm is a finite sequence of unambiguous instructions that accomplishes a well-defined task in a finite amount of time.
- Features Horowitz et al., 2008)
  - input (zero or more input values)
  - output (one or more output values)
  - definiteness (clear and unambiguous set of instructions)
  - termination (ends in a finite number of steps)
  - effectiveness (instructions must be feasible)

- Two common measures of efficiency (Cormen et al. (2009))
  - Space complexity: the amount of memory the algorithm needs
  - Time complexity: how fast the algorithm runs
- Big O notation
  - We define  $O(g(n)) = f(n)$  : There exist positive constants  $c$  and  $n_0$  such that  $f(n) \leq c \cdot g(n) \forall n \geq n_0$ . The expression  $f(n) = O(g(n))$  denotes the membership of  $f(n)$  in the set  $O(g(n))$  (Cormen et al., 2009).
  - E.g.,  $O(n)$  and  $O(\log n)$

## SPECIFYING ALGORITHMS

Algorithms need to be specified unambiguously. Methods of specifying algorithms include:

- Natural language
- Flowchart
- Pseudocode



## Natural language

- read the two numbers as input.
- let  $m$  be the maximum and  $n$  be the minimum of the two numbers.
- if  $n = 0$ , output  $m$  as the answer.
- otherwise, divide  $m$  by  $n$  and let  $r$  be the remainder. Now set  $m = n$  and  $n = r$  and return to the second step.

## Pseudocode

GCD

**begin**

**read**  $a, b$

$m \leftarrow \text{maximum}(a, b)$

$n \leftarrow \text{minimum}(a, b)$

**while**  $(n \neq 0)$

$r \leftarrow m \bmod n$

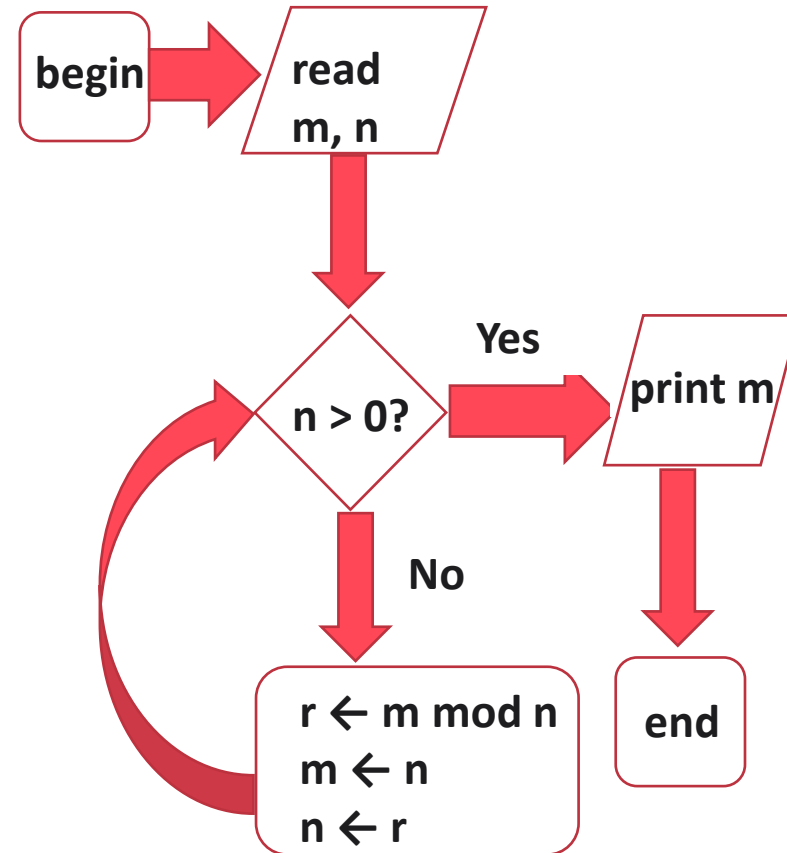
$m \leftarrow n, n \leftarrow r$

**endwhile**

**return**  $m$

**end**

## Flowchart



## DATA ENCAPSULATION

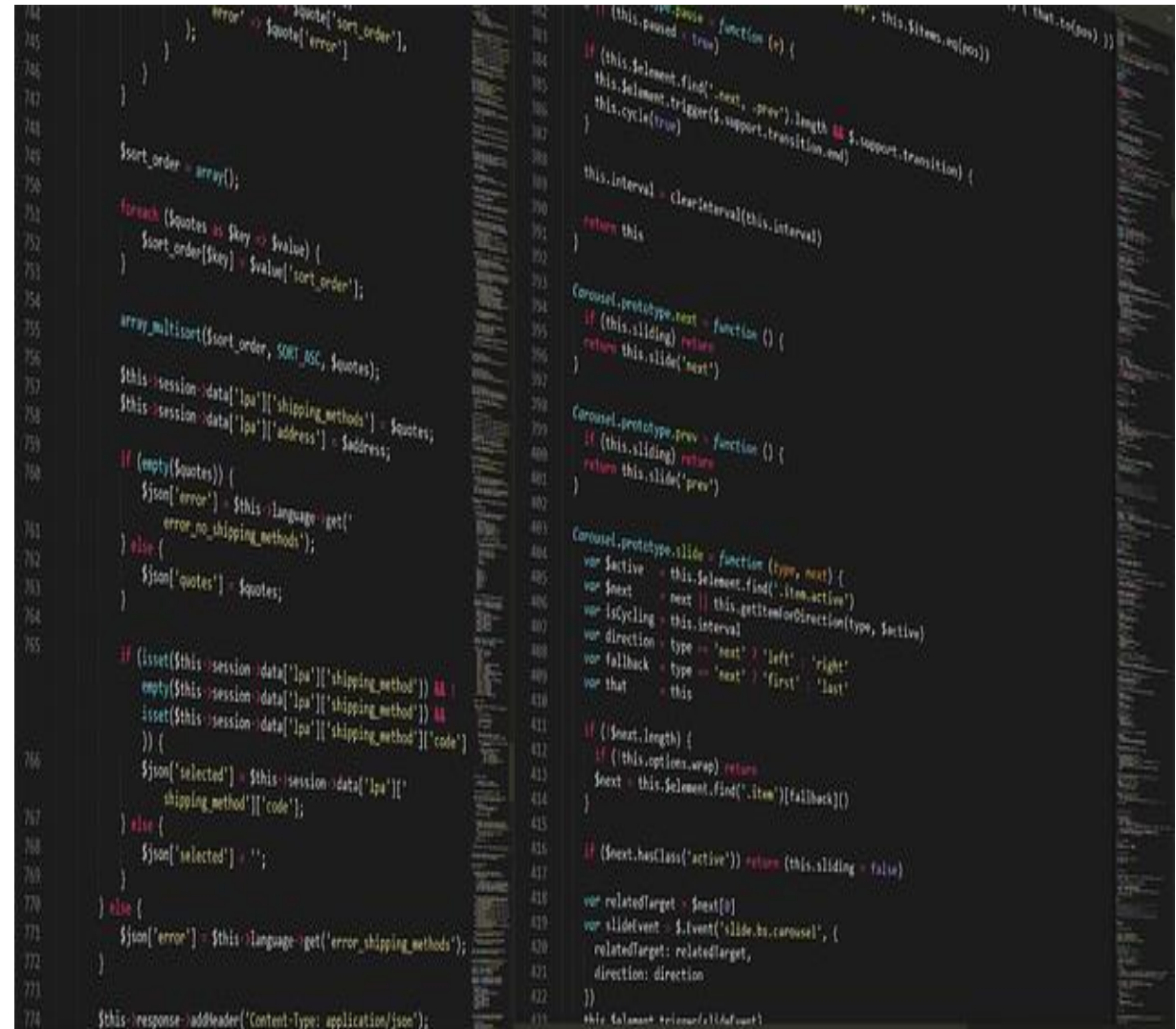
- Hiding data within an entity
- Methods to control access
- Data can be manipulated by a controlled set of defining operations.
- Only these operations depend on the internal representation.
- If the representation is updated, only the defining operations need to change.
- **Representation invariants** are enforced.
- Helps ensuring correct-by-construction implementation.



## ATTRIBUTES OF A GOOD LANGUAGE

Attributes of a good programming language identified over the years :

- Clarity, simplicity
- Expressivity
- Orthogonality
- Support for abstraction
- Portability
- Cost of use

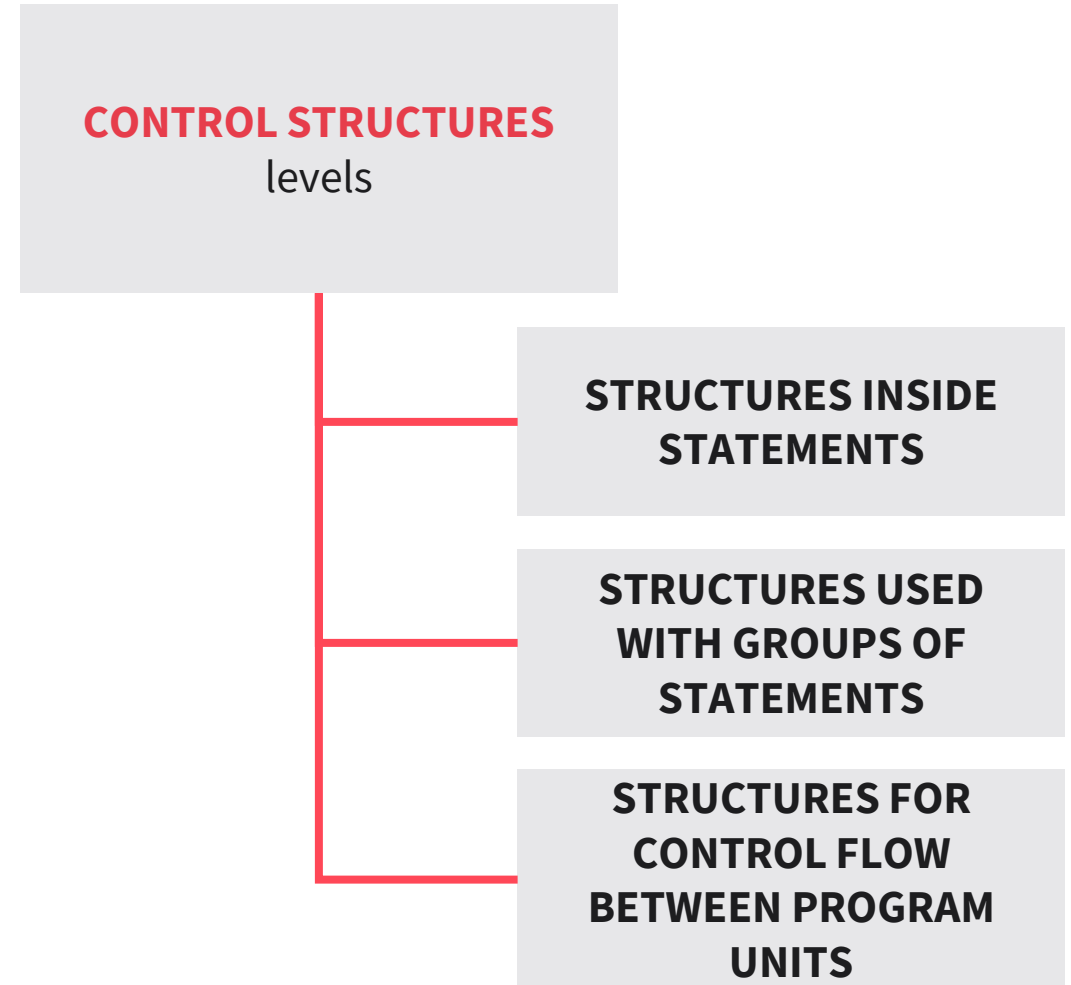


```
744 error: > $quote['sort_order'],  
745 };  
746 }  
747 }  
748  
749 $sort_order = array();  
750  
751 foreach ($quotes as $key => $value) {  
752     $sort_order[$key] = $value['sort_order'];  
753 }  
754  
755 array_multisort($sort_order, SORT_ASC, $quotes);  
756  
757 $this->session->data['lpa']['shipping_methods'] = $quotes;  
758 $this->session->data['lpa']['address'] = $address;  
759  
760 if (empty($quotes)) {  
761     $json['error'] = $this->language->get('error_no_shipping_methods');  
762 } else {  
763     $json['quotes'] = $quotes;  
764 }  
765  
766 if (isset($this->session->data['lpa']['shipping_method']) && !  
767     empty($this->session->data['lpa']['shipping_method']) &&  
768     isset($this->session->data['lpa']['shipping_method']['code'])  
769 ) {  
770     $json['selected'] = $this->session->data['lpa']['shipping_method']['code'];  
771 } else {  
772     $json['selected'] = '';  
773 }  
774  
775 if (isset($this->session->data['lpa']['shipping_method']) && !  
776     empty($this->session->data['lpa']['shipping_method']) &&  
777     isset($this->session->data['lpa']['shipping_method']['code'])  
778 ) {  
779     $json['selected'] = $this->session->data['lpa']['shipping_method']['code'];  
780 } else {  
781     $json['selected'] = '';  
782 }  
783  
784 $this->response->addHeader('Content-Type: application/json');  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```

## CONTROL STRUCTURES

- **Control flow** is the sequence in which program instructions are executed on a computer.
- **Control structures** facilitate the flow of control inside a program.
- Control structures operate at three levels.

Graphix X: Control Structures





## CONTROL STRUCTURES

Structures used  
inside  
statements

Arithmetic expressions, Boolean expressions

Assignments (compound, multiple assignments)

Comparison operators

Structures used  
with groups of  
statements

Conditional statements

Iterative loops like while, for

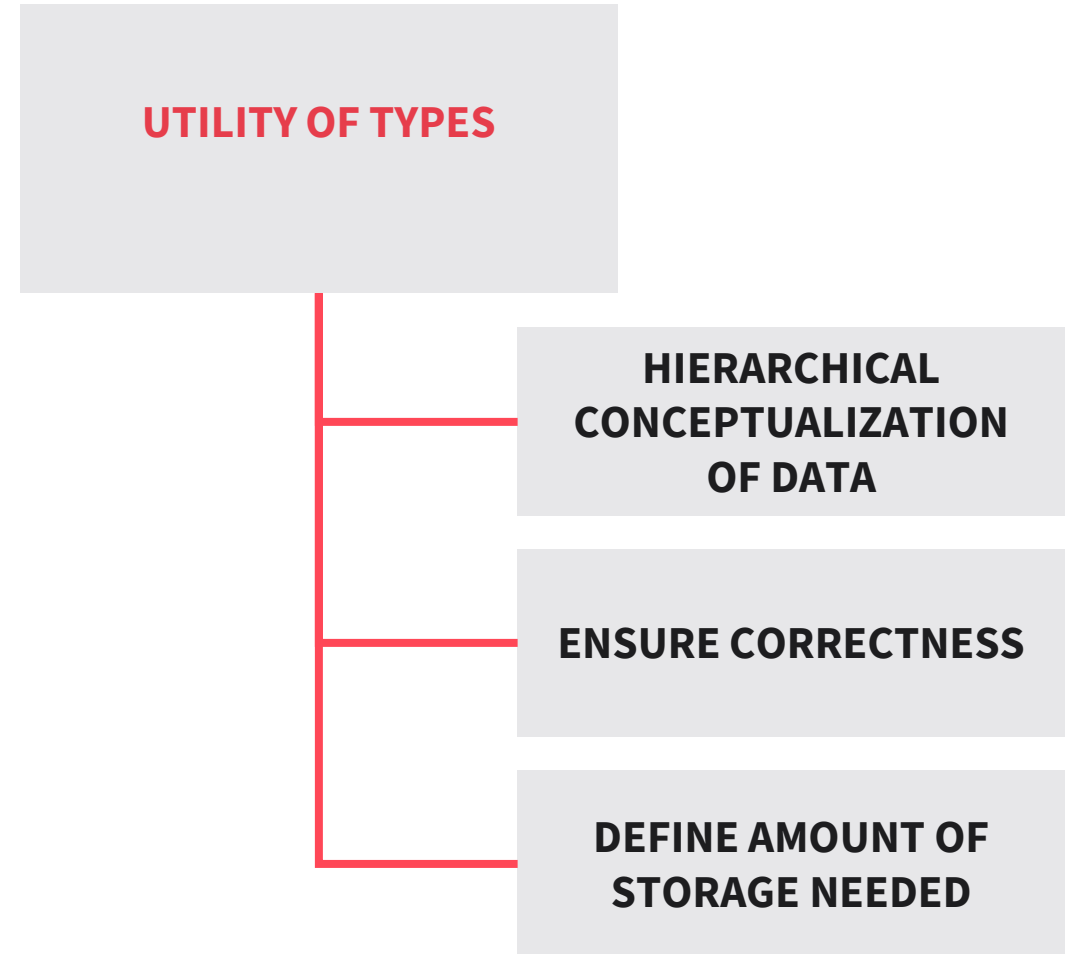
Structures that  
facilitate flow of  
control between  
program units

Function calls (built-in, user-defined), recursion

## TYPES

- In a programming language, a **type** is defined by a set of values and a set of operations.
- A **variable of a type** can only be operated on by operations defined.
- A type attaches specific meanings to an entity in a program like a variable. The hardware would not discriminate between meanings associated with a sequence of bits.

Graphix X: Utility of Types



**Type system:** logical system defined with a set of constructs to assign types to variables, expressions or return values of functions

- defines the set of built-in types for the language
- provides the constructs for defining new types
- defines rules for control of types
- defines type compatibility rules between expected and actual function parameters.
- defines rules for computing types of expressions

## BASIC DATA STRUCTURE LIST

- Also known as the singly linked list
- Unordered sequence of items
- Relative positions of the items are maintained.
- The location of the head of the list is explicitly known.
- The location of the  $(i+1)$ -th item in the sequence is stored with the  $i$ -th item.
- The last item has no next item.

### Operations supported by Linked List Data Structure

- `LinkedList()` constructs an empty list.
- `isEmpty()` returns True/False based on whether the list is empty or not.
- `getLength()` returns the number of elements in the list.
- `addNode(element)` adds a new element to the front of the list.
- `deleteNode(element)` removes the element from the list.
- `searchNode(element)` searches for the elements item in the list.

## BASIC DATA STRUCTURE CHAIN

- Also known as the doubly linked list
- Unordered sequence of items
- For each node, pointer to both predecessor and successor in the sequence are explicitly maintained.
- Extra space per node
- Both predecessor and successor information needs to be maintained during insert and delete operations.

Some operations supported by Chain Data Structure

- `DoublyLinkedList()` constructs an empty list.
- `isEmpty()` returns True/False based on whether the list is empty or not.
- `getLength()` returns the number of elements.
- `addNode(element, prev, next)` adds a new element between `prev` and `next`.
- `deleteNode(element)` removes the element from the list.
- `searchNode(element)` searches for the elements item in the list.

## BASIC DATA STRUCTURE TREES

- Fundamental data structure.
- Helps in representation of connectivity and hierarchy.
- Used for representing acyclic relationships connecting entities.
- A rooted tree is a **recursive** structure.
- A root node is connected to child nodes which are roots of subtrees.



A family tree



- Explain the role of algorithms, data structures and programming languages in programming.
- Represent algorithms in various ways.
- Understand the role of abstraction and encapsulation in programming.
- Define concepts of control structures in programming languages.
- Differentiate between different data types.
- Create basic data structures: List, Chain, Tree.

**SESSION 1**

# **TRANSFER TASK**



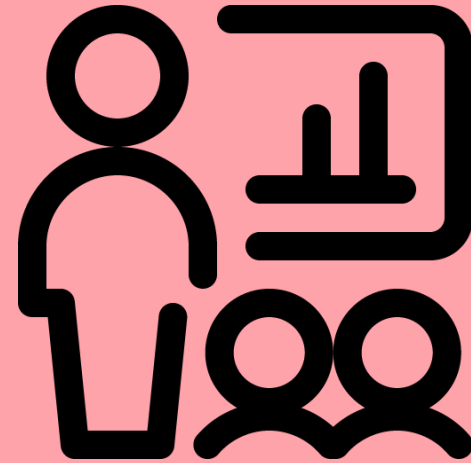
## TRANSFER TASKS

1. If the input to a program is a set of IDs for employees working in a company, what are the advantages of arranging the data as a tree as opposed to a singly-linked list?
2. Describe examples of two queries that may be easily supported by such a tree.

TRANSFER TASK  
PRESENTATION OF THE RESULTS

Please present your  
results.

The results will be  
discussed in plenary.





1. If the defining operations in a program can only generate objects that follow certain rules, we can obtain accurate results by implementing a correct-by-construction approach. What are these rules called?
  - a) representation invariants.
  - b) defining rules.
  - c) construction rules.
  - d) operating rules.



2. In an expression, the order of operations may depend on which of the following?
- a) rules for operator precedence only.
  - b) rules for associativity only.
  - c) rules for operator precedence and rules for associativity.
  - d) neither rules for operator precedence nor rules for associativity.



3. In a linked list of integers, the operations of which take  $O(n)$  time, assuming all operations have a single parameter “key,” which is the value to be added, deleted, or searched for?
- a) searchNode and addNode
  - b) deleteNode and searchNode
  - c) addNode and deleteNode
  - d) searchNode only

## LIST OF SOURCES

Cormen, T.H., Leiserson, C.E., Rivest, R.L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

Goodrich, M.T., Tamassia, R., & Goldwasser, M.H. (2013). *Data Structures and Algorithms in Python*. Wiley.

Pratt, T.W., & Zelkowitz, M.V. (2001). *Programming Languages: Design and Implementation* (4th ed.). Prentice-Hall.

Sebesta, R.W. (2016). *Concepts of Programming Languages* (11th ed.). Pearson.

© 2022 IU Internationale Hochschule GmbH

This content is protected by copyright. All rights reserved.

This content may not be reproduced and/or electronically edited, duplicated, or distributed in any kind of form without written permission by the IU Internationale Hochschule GmbH.