
GHENT
UNIVERSITY

FACULTY OF ECONOMICS AND BUSINESS ADMINISTRATION

DEPARTMENT BUSINESS INFORMATICS AND OPERATIONS MANAGEMENT
RESEARCH GROUP BUSINESS INFORMATICS

INTRODUCTION TO PROGRAMMING

Chapter 0

GHENT UNIVERSITY

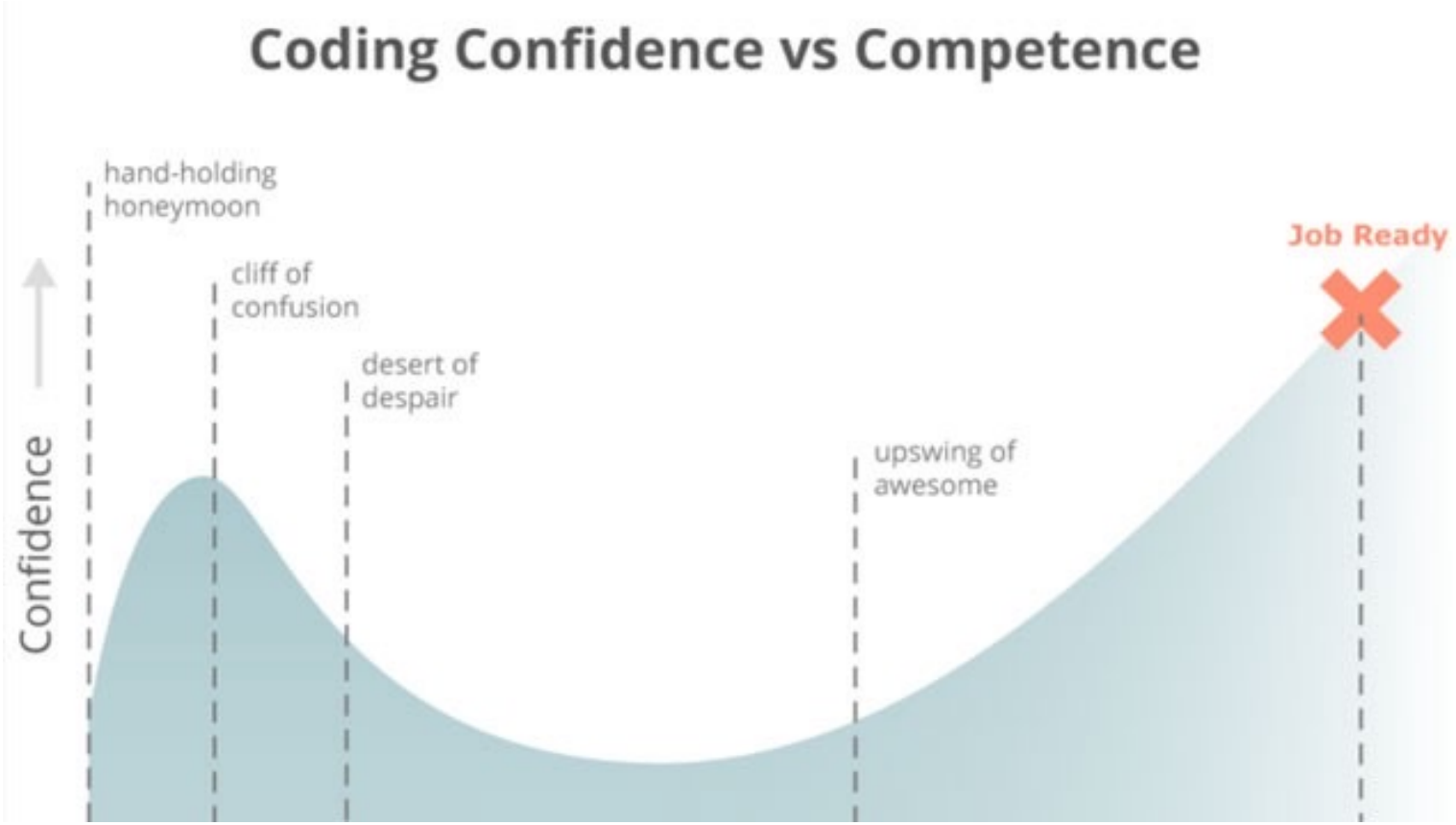
CONTENT

- Why ?
  - Programming
  - Python ?
  - Python Programming for Economics Students?
- Installation Python and PyCharm
- First Python Program
- Interactive session
- Parts of a program
- Developing an Algorithm
- Errors

WHY PROGRAMMING

- Problem Solving: ability to break down complex problem into smaller solvable parts
- Creativity: bring your ideas to live
- Career Opportunities: high demand for programming skills

WHY PROGRAMMING



GHENT UNIVERSITY

PYTHON

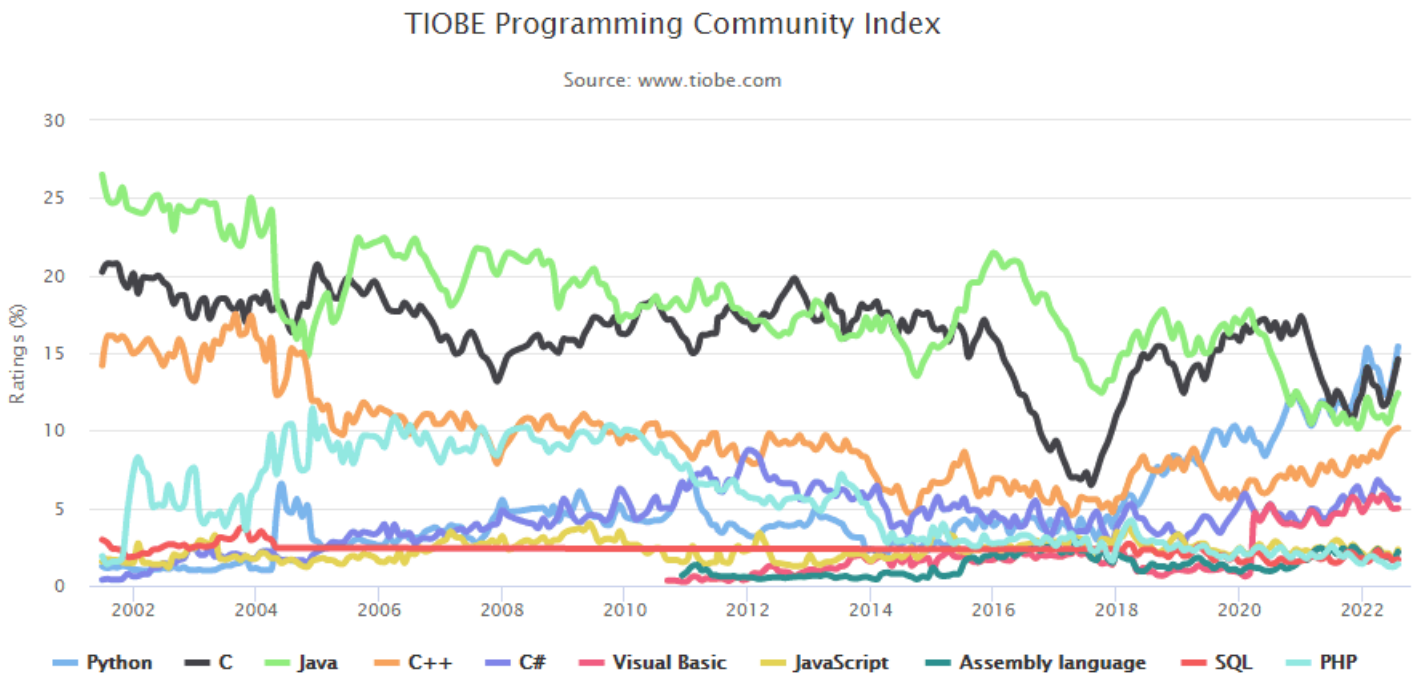
- Guido van Rossum (NL)
- Monty Python's Flying Circus
- Python is
  - General-purpose programming language
  - Interpreted
  - Object oriented
- Two versions: Python 2 and Python 3

WHY PYTHON?

- Low “cognitive load” for the student
  - In python one - and preferably only one - obvious way to do something
- Easy to apply to problems you encounter in real world
  - Python is a best practices languages
  - Python is a “batteries included” language
- Broad support across many disciplines
  - Python is Open Source

WHY PYTHON

- Other important Programming Languages to Learn in 2023:
  - Javascript
  - Java
  - Typescript
  - C/C++
  - C#
  - SQL
  - PHP
  - Visual Basic
  - Go



GHENT UNIVERSITY

PYTHON PROGRAMMING FOR BUSINESS ECONOMICS STUDENTS?

- Data Analysis: Pandas and Numpy can be used to clean, manipulate and analyze large data sets
- Simulation: build simulation models for economic models
- Econometrics: Statsmodels and Scipy can be used to perform lineair and non-linear regression
- Visualisation: Matplotlib and Seaborn can be used to create visualisation of economic data
- Web scraping: extract desired information from websites and store it an structured format
- Automation: automate repetitive tasks

INITIAL COMPETENCES: INFORMATICS

- Programming languages
  - Variables and data types
  - Data structures
  - Arrays
  - Constants and literals
  - Assignment statements
  - Control Flow
  - Functions

INITIAL COMPETENCES: INFORMATICS

- Data Abstractions
  - Data structures
  - Pointers
  - Classes and Objects
  - ….

GHENT UNIVERSITY

INSTALLATION PYTHON AND IDE

- See ufora webpage
- Python 3.12
- Use PyCharm or Visual Studio Code as IDE
- BYOD

  ➔ install Python and IDE on your own device

TOPICS

- Basics programming with python
- Control structures: selection, iteration and functions
- Data structures: Strings, Lists, Tuples, Dictionaries and Sets
- (Files and Exceptions)
- Intro object-oriented programming

COURSE MATERIAL

- Introduction to Python Programming and Data Structures, Third Edition, Y. Daniel Lang
- Slides available via Ufora
- IDE

COURSE MATERIAL

- Dodona.ugent.be
  - Online platform that gives feedback on programming exercises
  - Use UGent account to access
  - Aanmelden voor dodona cursus via: https://dodona.be/nl/courses/4238/?secret=qZsQd
  - Deadlines assignments

TEACHING METHOD: ON CAMPUS LECTURE

- Weekly Monday 4 o’clock, Aud Devreker
- Lectures are recorded
- BYOD

TEACHING METHODS: ONLINE SESSIONS

- Weekly Monday 8.30
- Online via MS Teams
- Exercises available via Ufora
- Respond college

TEACHING METHODS: DODONA ASSIGNMENTS

- Tasks are evaluated as follows:
  - 0.5/0.5 if 3 out of 4 learning activities were submitted correctly
  - 0.25/0.5 if 2 out of 4 learning activities were submitted correctly
  - 0/0.5 if fewer than 2 out of 4 learning activities were submitted correctly
- **Note:** If Dodona indicates that the code is correct, this does not necessarily mean that the maximum score is automatically achieved for the exercises. Additional tests after submission are possible.

PLANNING

Week	Date	Time	What	Content
1	23/09/2024	8.30 ➔ 9.45	NO Lecture	
	23/09/2024	16.00 ➔ 17.15	On campus lecture Aud Devreker	Intro
2	30/09/2024	8.30 ➔ 9.45	Online session	First program + Dodona + IDE
	30/09/2024	16.00 ➔ 17.15	On campus lecture Aud Devreker	Elementary Programming
3	07/10/2024	8.30 ➔ 9.45	Online session	
	07/10/2024	16.00 ➔ 17.15	On campus lecture Aud Devreker	Selections
	13/10/2024	23.59	DEADLINE ASSIGNMENT 1	

PLANNING

Week	Date	Time	What	Content
4	14/10/2024	8.30 ➔ 9.45	Online Session	
	14/10/2024	16.00 ➔ 17.15	On campus lecture Aud Devreker	Mathematical Functions, Strings and Objects
5	21/10/2024	8.30 ➔ 9.45	Online session	
	21/10/2024	16.00 ➔ 17.15	On campus lecture Aud Devreker	Loops
6	28/10/2024	8.30 ➔ 9.45	NO LECTURE	
	28/10/2024	16.00 ➔ 17.15	NO LECTURE	
	03/11/2024	23.59	DEADLINE ASSIGNMENT 2	

PLANNING

Week	Date	Time	What	Content
7	04/11/2024		Online lecture	
	04/11/2024	16.00 ➔ 17.15	On campus lecture Aud Devreker	Functions
8	11/11/2024	8.30 ➔ 9.45	NO LECTURE	
	11/11/2024	16.00 ➔ 17.15	NO LECTURE	
9	18/11/2024	8.30 ➔ 9.45	Online lecture	
	18/11/2024	16.00 ➔ 17.15	On campus lecture Aud Devreker	Lists
	24/11/2024	23.59	Deadline Assignment 3	

PLANNING

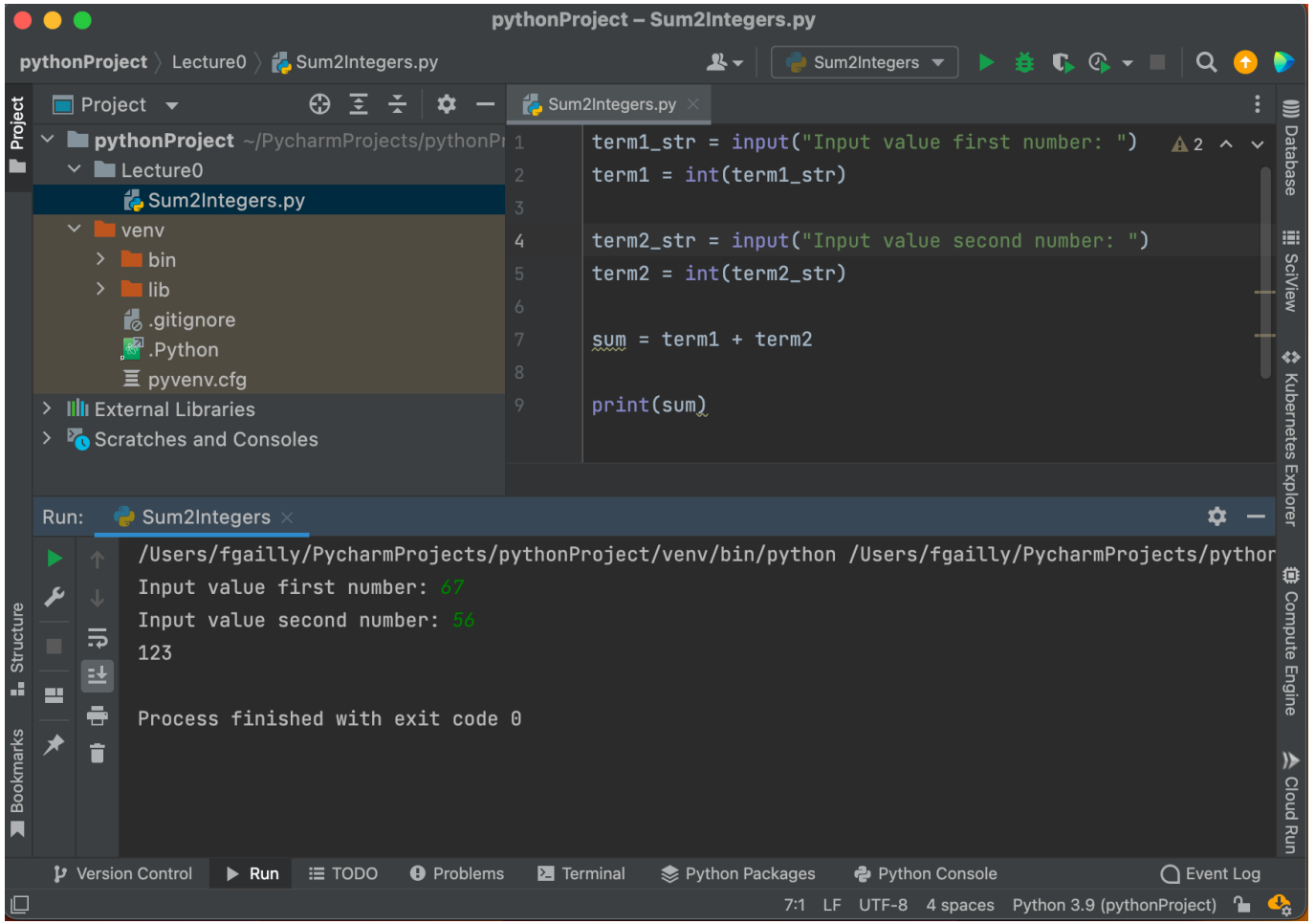
Week	Date	Time	What	Content
10	25/11/2024	8.30 ➔ 9.45	Online lecture	
	25/11/2024	16.00 ➔ 17.15	On campus lecture Aud Devreker	Objects and classes
11	02/12/2024	8.30 ➔ 9.45	Online lecture	
	02/12/2024	16.00 ➔ 17.15	On campus lecture Aud Devreker	Inheritance and Polymorphism
12	09/12/2024	8.30 ➔ 9.45	Online werkcollege	Optional
	09/12/2024	16.00 ➔ 17.15	On campus lecture Aud Devreker	Optional
	15/12/2024	23.59	Deadline Taak 4	

EVALUATION:

- Continuous assessment:
  - Dodona assignment ➔ 10% or 2 points out of 20
- End-of-term assessment
  - Part 1: 10 multiple choice questions
    - 45 % or 9 points out of 20
    - Closed book
  - Part 2: Programming exercise
    - 45% or 9 points out of 20
    - Submit via Dodona

FIRST PYTHON PROGRAM

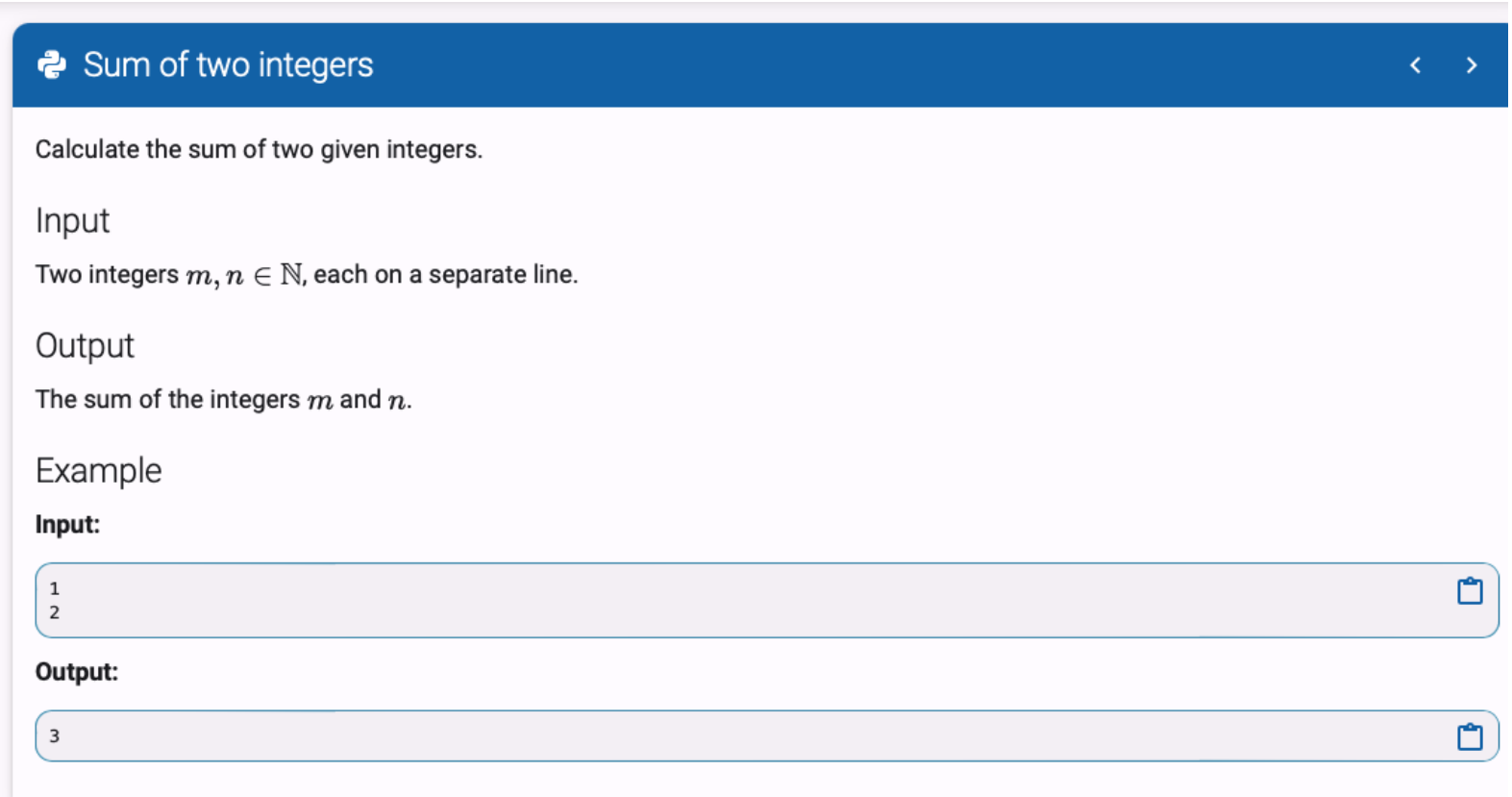
– sum of two integers



GHENT UNIVERSITY

FIRST PYTHON PROGRAM

- Check program in Dodona



GHENT UNIVERSITY

INTERACTIVE SESSION

- Python in an interpreted language
- Interactive session in console
- Interactive session in PyCharm
- Task: Execute first program as an Interactive session

PARTS OF A PROGRAM

- Modules
- Statements and expressions
- Whitespace
- Comments
- Special Python elements: tokens
- Naming objects
- Recommendation on Naming

PARTS OF A PROGRAM

- Modules:
  - Contains a set of python commands
  - Can be stored as a file
  - Can be imported:

```
import [module] #load the module
```

- Expressions and statements
  - Expression: Combination of values and operations that creates a new value
  - Statement: Does not return a value but does perform some task

PARTS OF A PROGRAM

- Whitespace:
  - Space, tab, return, linefeed, formfeed, vertical tab
  - Whitespace is ignored within expressions and statements
  - Indentation plays special role in Python:
    - Make code readable
    - Required by python for grouping
  - Continuation:
    - Splitting long lines of code using backslash

PARTS OF A PROGRAM

- Comments:
  - Comments improve readability
  - Used character: #
- Special Python elements: tokens
  - Keywords: and, while, print, import, …
  - Operators: +, -, *, **, ==, +=, !=, …
  - Punctuators and Delimiters: (,),”,’,…
  - Literals: 123, a, true

PARTS OF A PROGRAM

- Naming objects:
  - Every name must begin with letter or underscore
  - After the first letter, any combination of letters, numbers and underscores
  - Name cannot be a keyword
  - Name cannot have delimiters, punctuations or operators
  - Uppercase is different from lowercase

PARTS OF A PROGRAM

- Recommendation on Naming:
  - Google style guide for pyth:
    https://google.github.io/styleguide/pyguide.html
  - Will be introduced as we go along

DEVELOPING AN ALGORITHM

- Algorithm: A method – a sequence of steps – that describes how to solve a problem or class of problems
- Can be represented in different ways
- Test your code !!!!

PROGRAMMING ERRORS

- Syntax Errors
  Error in code construction
- Runtime Errors
  Causes the program to abort
- Logic Errors
  Produces incorrect result

FACULTY OF ECONOMICS AND BUSINESS ADMINISTRATION

Frederik Gailly

BUSINESS INFORMATICS RESEARCH GROUP

E frederik.gaily@ugent.be

www.ugent.businessinformatics.be

GHENT UNIVERSITY

FACULTEIT ECONOMIE
EN BEDRIJFSKUNDE

ELEMENTARY PROGRAMMING

Chapter 2

UNIVERSITEIT
GENT

MOTIVATIONS

Suppose, for example, that you need to take out a student loan. Given the loan amount, loan term, and annual interest rate, can you write a program to compute the monthly payment and total payment? This chapter shows you how to write programs like this. Along the way, you learn the basic steps that go into analyzing a problem, designing a solution, and implementing the solution by creating a program.

OBJECTIVES

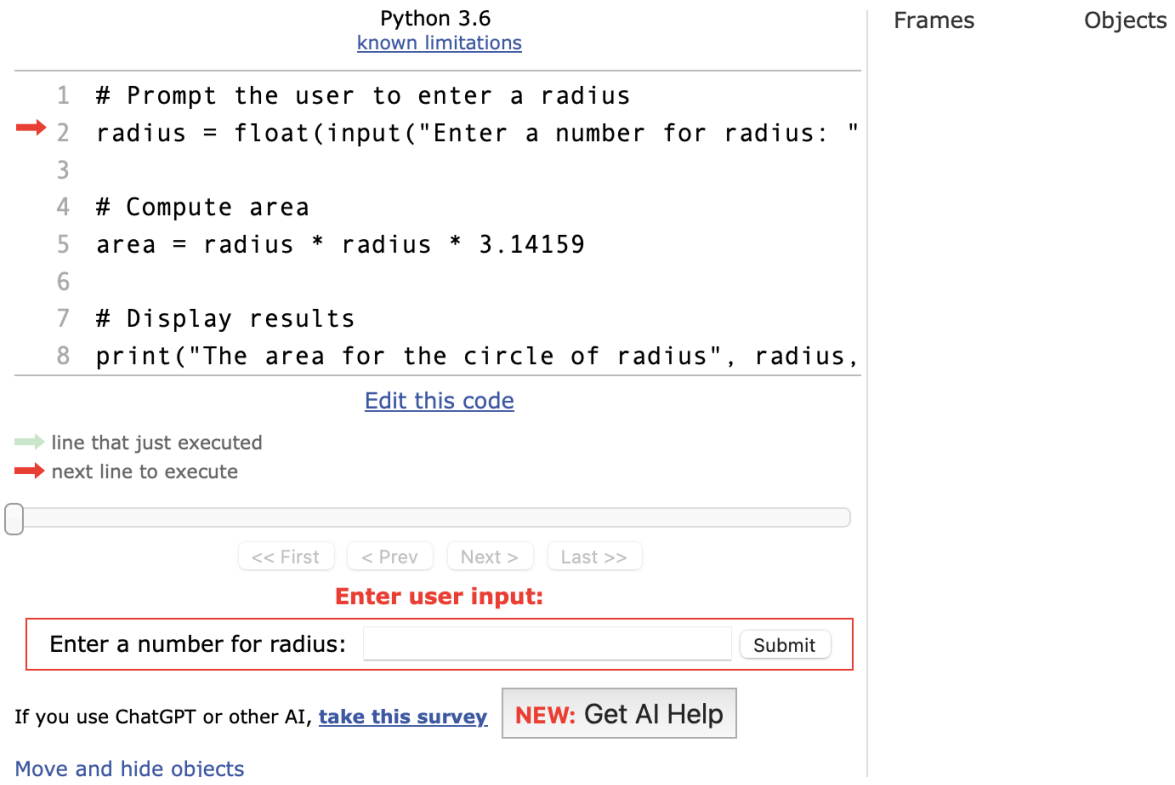
- To write programs that perform simple computations.
- To obtain input from a program's user by using the input function.
- To use identifiers to name variables.
- To assign data to variables.
- To define named constants.
- To use the operators +, -, *, /, //, %, and **.
- To write and evaluate numeric expressions.
- To use augmented assignment operators to simplify coding
- To perform numeric type conversion and rounding with the int and round functions
- To obtain the current system time by using time.time()).
- To describe the software development process and apply it to develop the loan payment program
- To compute and display the distance between two points.

INTRODUCING PROGRAMMING WITH AN EXAMPLE

- Computing the Area of a Circle
  - Get the Circle Radius from the user
  - Compute Area using formula

    area = radius * radius * PI
  - Display the result

TRACE A PROGRAM EXECUTION



UNIVERSITEIT GENT

IDENTIFIERS

- An identifier is a sequence of characters that consists of letters, digits, underscores (_), and asterisk (*).
- An identifier must start with a letter or an underscore. It cannot start with a digit.
- An identifier cannot be a reserved word. (See Appendix A, "Python Keywords," for a list of reserved words.) Reserved words have special meanings in Python, which we will later discuss.
- An identifier can be of any length.

VARIABLES, ASSIGNMENT STATEMENTS AND EXPRESSIONS

- Variables
  - Radius and Area are variables
  - Variables have a value
  - When you declare a variable you do not need to specify a type
  - Python determines type based on value

VARIABLES, ASSIGNMENT STATEMENTS AND EXPRESSIONS

- Assignment Statements
  - Values are assigned to variables using assignment statement
  - area = radius * radius * 3.1419
- Expression
  - Computation involving values, variables and operators

SIMULTANEOUS ASSIGNMENT

var1, var2, ..., varn = exp1, exp2, ..., expn

```
1   # Prompt the user to enter three numbers
2   number1, number2, number3 = eval(input(
3     "Enter three numbers separated by commas: "))
4
5   # Compute average
6   average = (number1 + number2 + number3) / 3
7
8   # Display result
9   print("The average of", number1, number2, number3,
10      "is", average)
11
```

UNIVERSITEIT GENT

NAMED CONSTANTS

The value of a variable may change during the execution of a program, but a named constant or simply constant represents permanent data that never changes. Python does not have a special syntax for naming constants. You can simply create a variable to denote a constant. To distinguish a constant from a variable, use all uppercase letters to name a constant.

NUMERICAL DATA TYPES

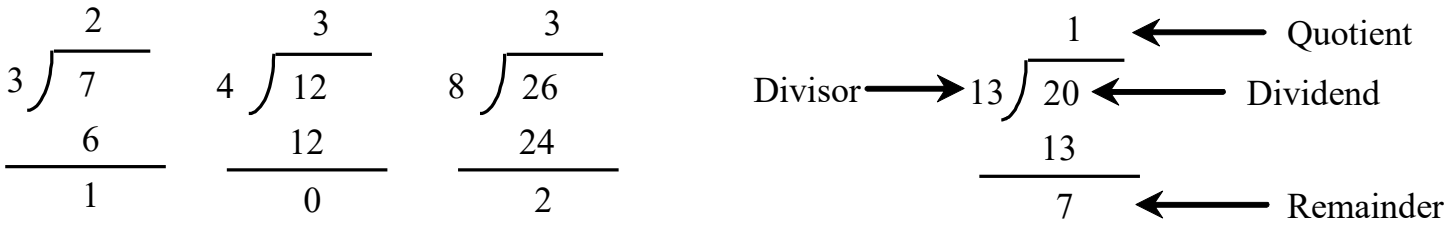
- integer: e.g., 3, 4
- float: e.g., 3.0, 4.0

NUMERIC OPERATORS

Name	Meaning	Example	Result
+	Addition	34 + 1	35
-	Subtraction	34.0 - 0.1	33.9
*	Multiplication	300 * 30	9000
/	Float Division	1 / 2	0.5
//	Integer Division	1 // 2	0
**	Exponentiation	4 ** 0.5	2.0
%	Remainder	20 % 3	2

THE % OPERATOR

- Remainder operator
- Very useful in programming
  - number % 2 equals 0 ➔ even number
  - Number % 2 equals 1 ➔ odd number



UNIVERSITEIT GENT

PROBLEM: DISPLAYING TIME

– Write a program that obtains minutes and remaining seconds from seconds.

```
1  # Prompt the user for input
2  seconds = int(input("Enter an integer for seconds: "))
3
4  # Get minutes and remaining seconds
5  minutes = seconds // 60     # Find minutes in seconds
6  remainingSeconds = seconds % 60   # Seconds remaining
7  print(seconds, "seconds is", minutes,
8      "minutes and", remainingSeconds, "seconds")
9
```

UNIVERSITEIT GENT

OVERFLOW

- When a variable is assigned a value that is too large (in size) to be stored, it causes overflow. For example, executing the following statement causes overflow.

```
>>> 245.0 ** 1000
OverflowError: 'Result too large'
```

UNDERFLOW

- When a floating-point number is too small (i.e., too close to zero) to be stored, it causes underflow. Python approximates it to zero. So normally you should not be concerned with underflow.

SCIENTIFIC NOTATION

- Floating-point literals can also be specified in scientific notation, for example, 1.23456e+2, same as 1.23456e2, is equivalent to 123.456, and 1.23456e-2 is equivalent to 0.0123456.
- E (or e) represents an exponent and it can be either in lowercase or uppercase.

ARITHMETIC EXPRESSIONS

$$\frac{3+4x}{5} - \frac{10(y-5)(a+b+c)}{x} + 9\left(\frac{4}{x} + \frac{9+x}{y}\right)$$

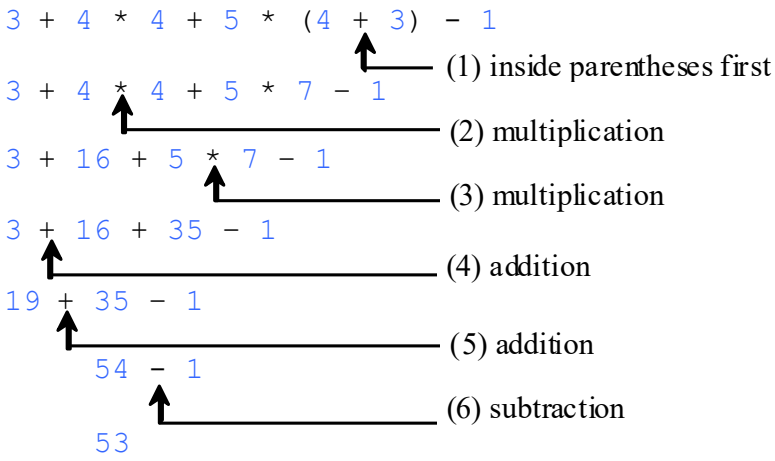
is translated to

```
(3+4*x)/5 – 10*(y-5)*(a+b+c)/x + 9*(4/x + (9+x)/y)
```

UNIVERSITEIT
GENT

HOW TO EVALUATE AN EXPRESSION

Though Python has its own way to evaluate an expression behind the scene, the result of a Python expression and its corresponding arithmetic expression are the same. Therefore, you can safely apply the arithmetic rule for evaluating a Python expression.



UNIVERSITEIT GENT

AUGMENTED ASSIGNMENT OPERATORS

Operator	*Example*	*Equivalent*
`+=`	`i += 8`	`i = i + 8`
`-=`	`f -= 8.0`	`f = f - 8.0`
`*=`	`i *= 8`	`i = i * 8`
`/=`	`i /= 8`	`i = i / 8`
`//=`	`i //= 8`	`i = i // 8`
`%=`	`i %= 8`	`i = i % 8`
`**=`	`i **= 8`	`i = i ** 8`

UNIVERSITEIT GENT

TYPE CONVERSION AND ROUNDING

- datatype(value)

- Examples
  - int(4.5) => 4
  - float(4) => 4.0

PROBLEM: KEEPING TWO DIGITS AFTER DECIMAL POINTS

Write a program that displays the sales tax with two digits after the decimal point.

```
1  # Prompt the user for input
2  purchaseAmount = float(input("Enter purchase amount: "))
3
4  # Compute sales tax
5  tax = purchaseAmount * 0.06
6
7  # Display tax amount with two digits after decimal point
8  print("Sales tax is", int(tax * 100) / 100)
```

UNIVERSITEIT GENT

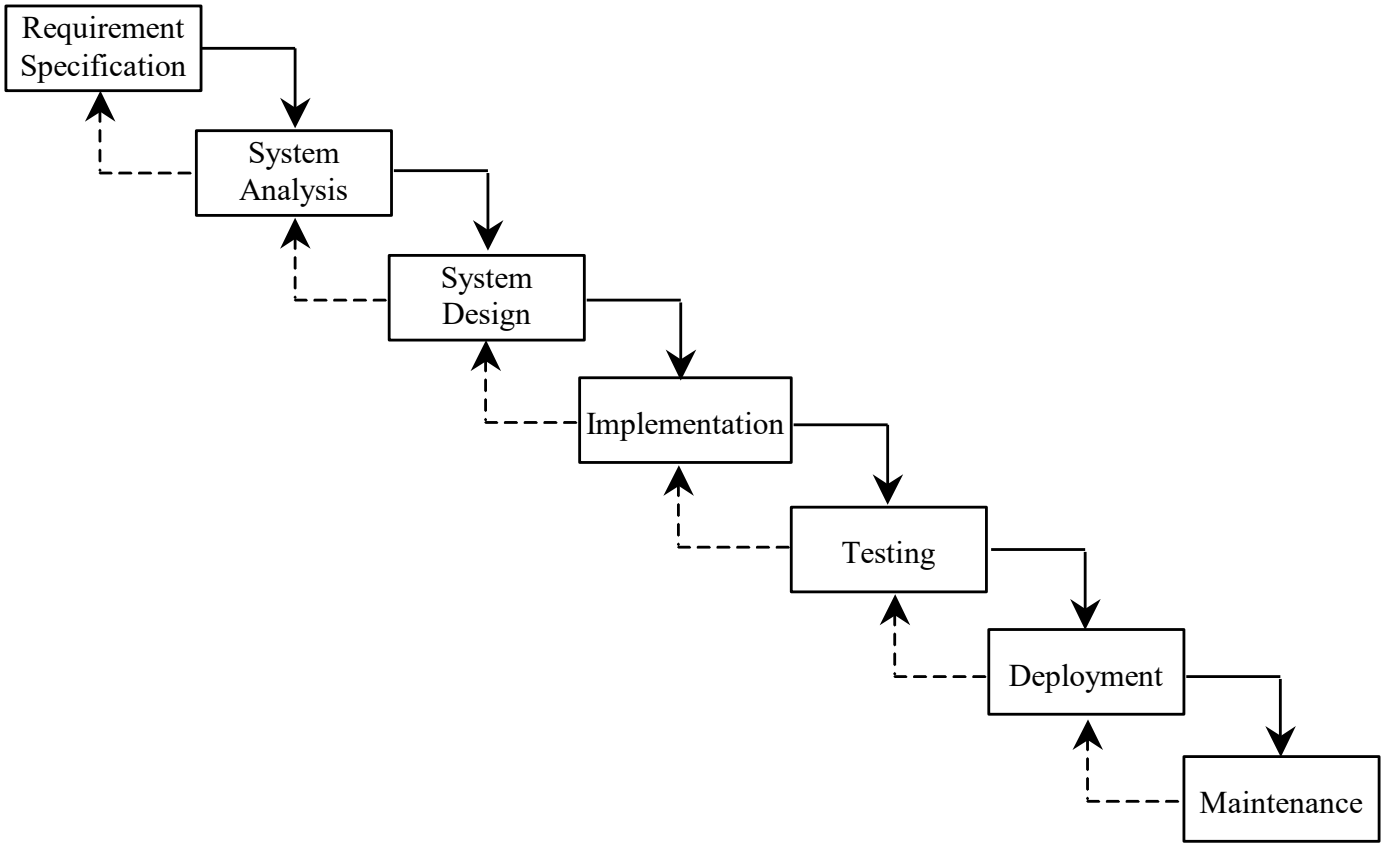
PROBLEM: DISPLAYING CURRENT TIME

Write a program that displays current time in GMT in the format hour:minute:second such as 1:45:19.

The time.time() function returns the current time in seconds with millisecond precision since the midnight, January 1, 1970 GMT. (1970 was the year when the Unix operating system was formally introduced.) You can use this function to obtain the current time, and then compute the current second, minute, and hour as follows.

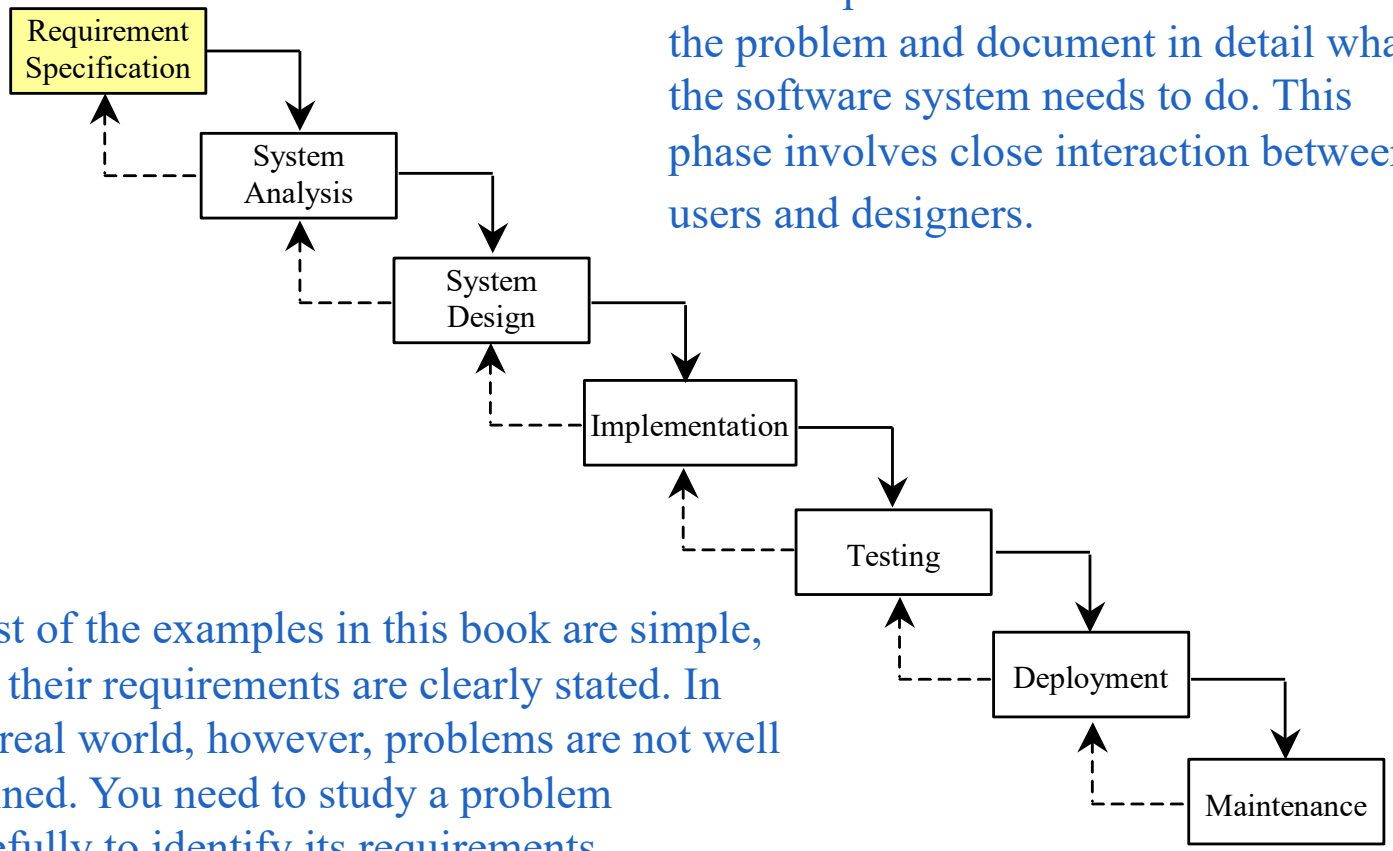
ShowCurrentTime

SOFTWARE DEVELOPMENT PROCESS



UNIVERSITEIT GENT

REQUIREMENT SPECIFICATION

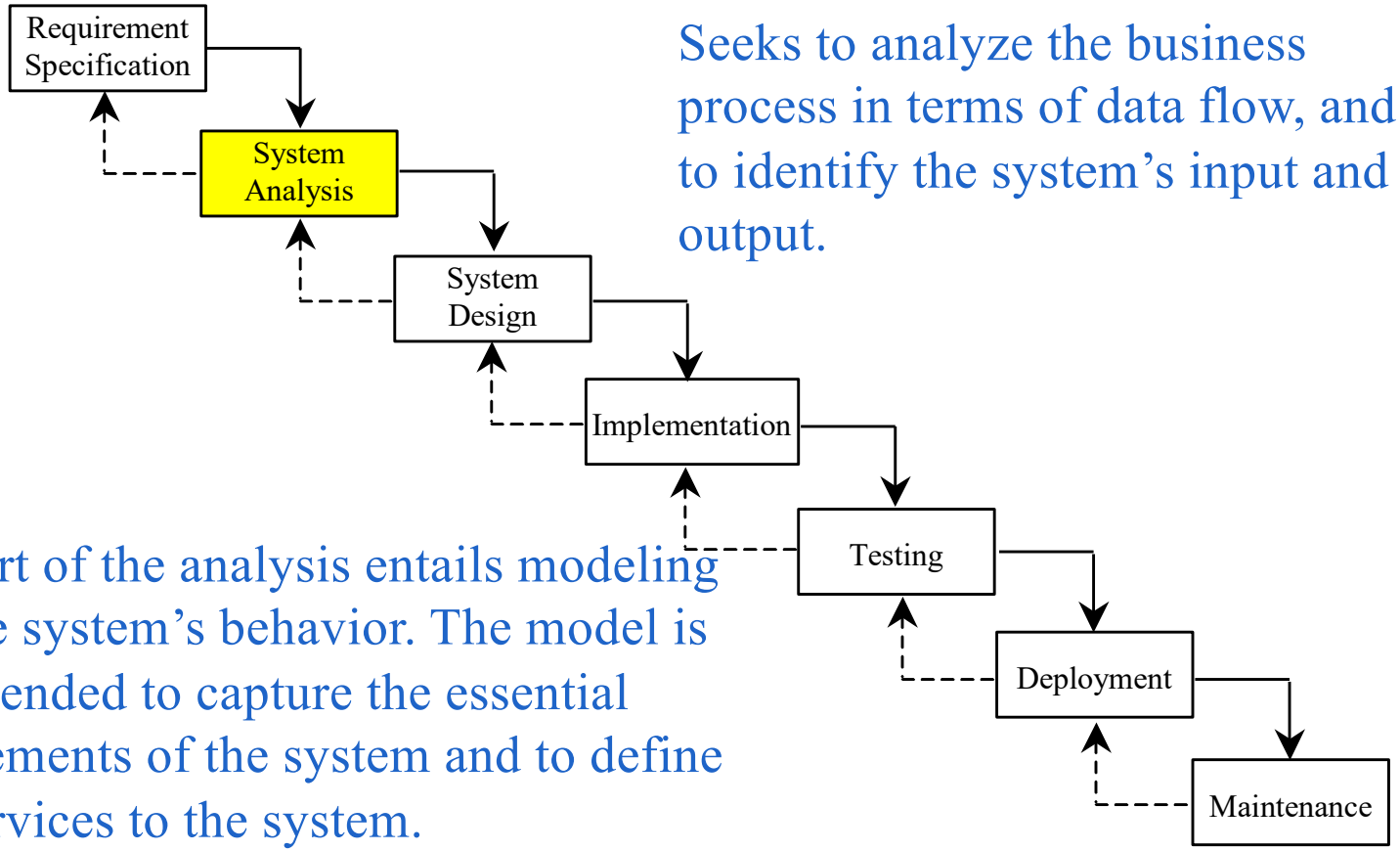


A formal process that seeks to understand the problem and document in detail what the software system needs to do. This phase involves close interaction between users and designers.

Most of the examples in this book are simple, and their requirements are clearly stated. In the real world, however, problems are not well defined. You need to study a problem carefully to identify its requirements.

UNIVERSITEIT GENT

SYSTEM ANALYSIS

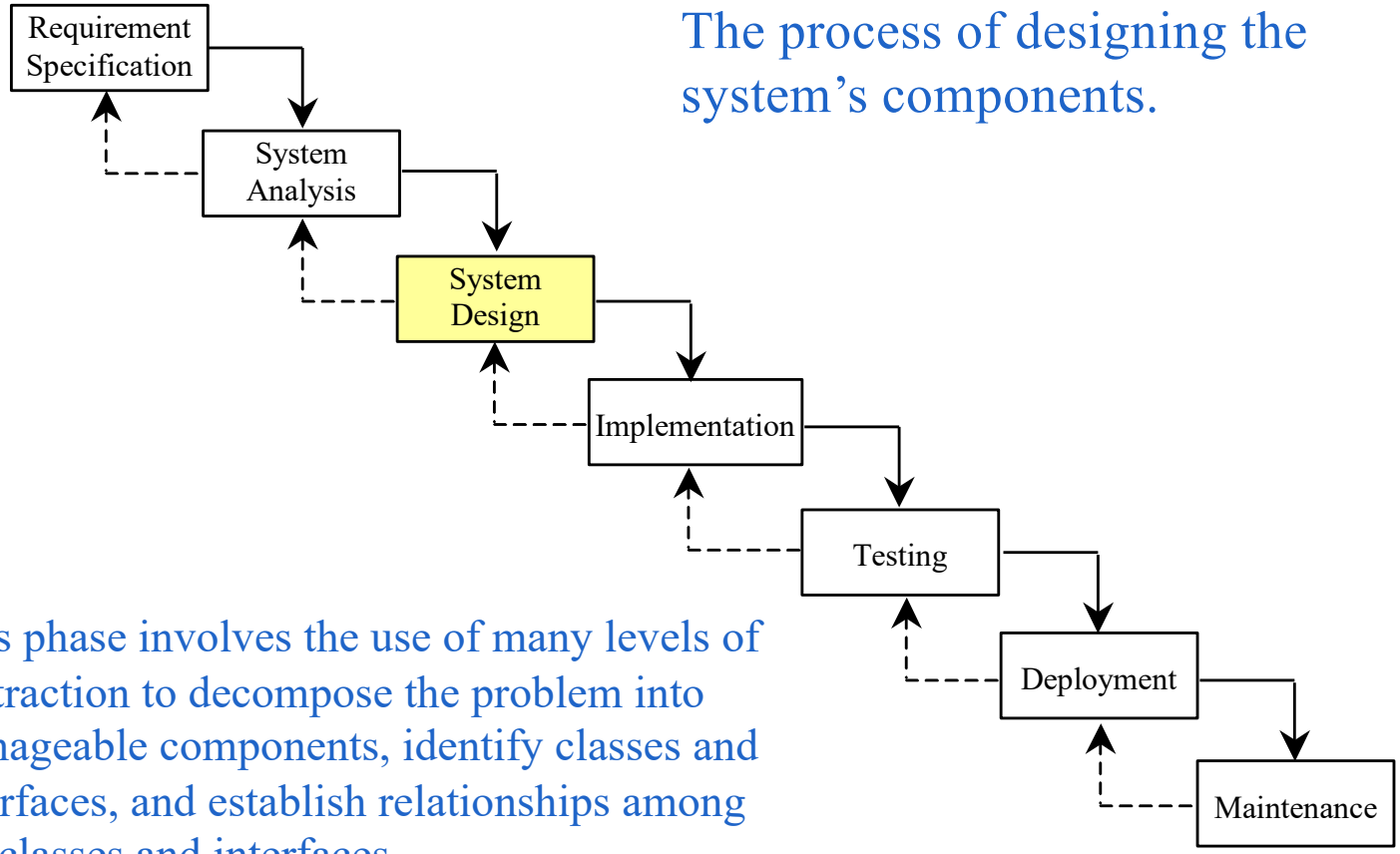


Seeks to analyze the business process in terms of data flow, and to identify the system's input and output.

Part of the analysis entails modeling the system's behavior. The model is intended to capture the essential elements of the system and to define services to the system.

UNIVERSITEIT GENT

SYSTEM DESIGN

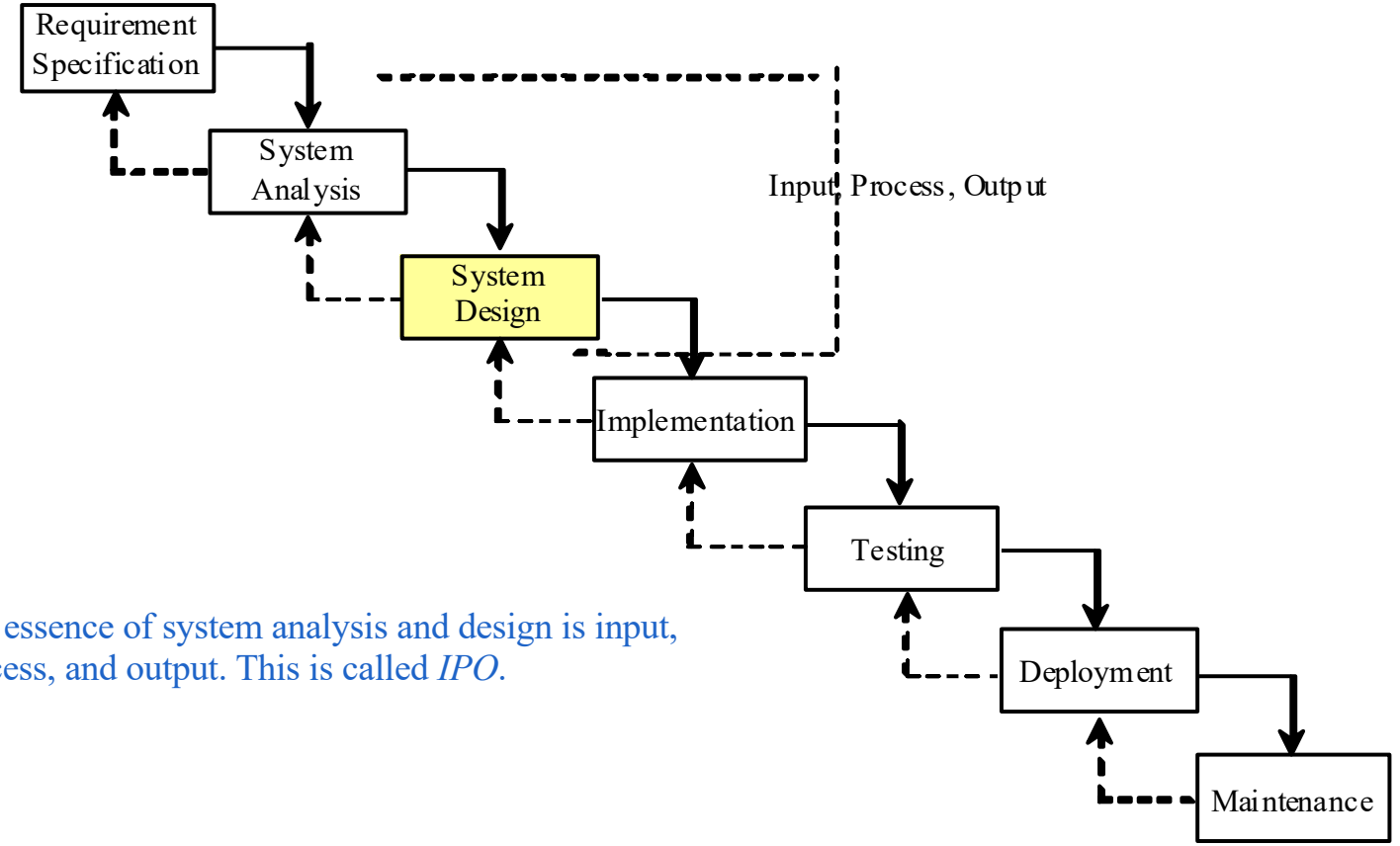


The process of designing the system's components.

This phase involves the use of many levels of abstraction to decompose the problem into manageable components, identify classes and interfaces, and establish relationships among the classes and interfaces.

UNIVERSITEIT GENT

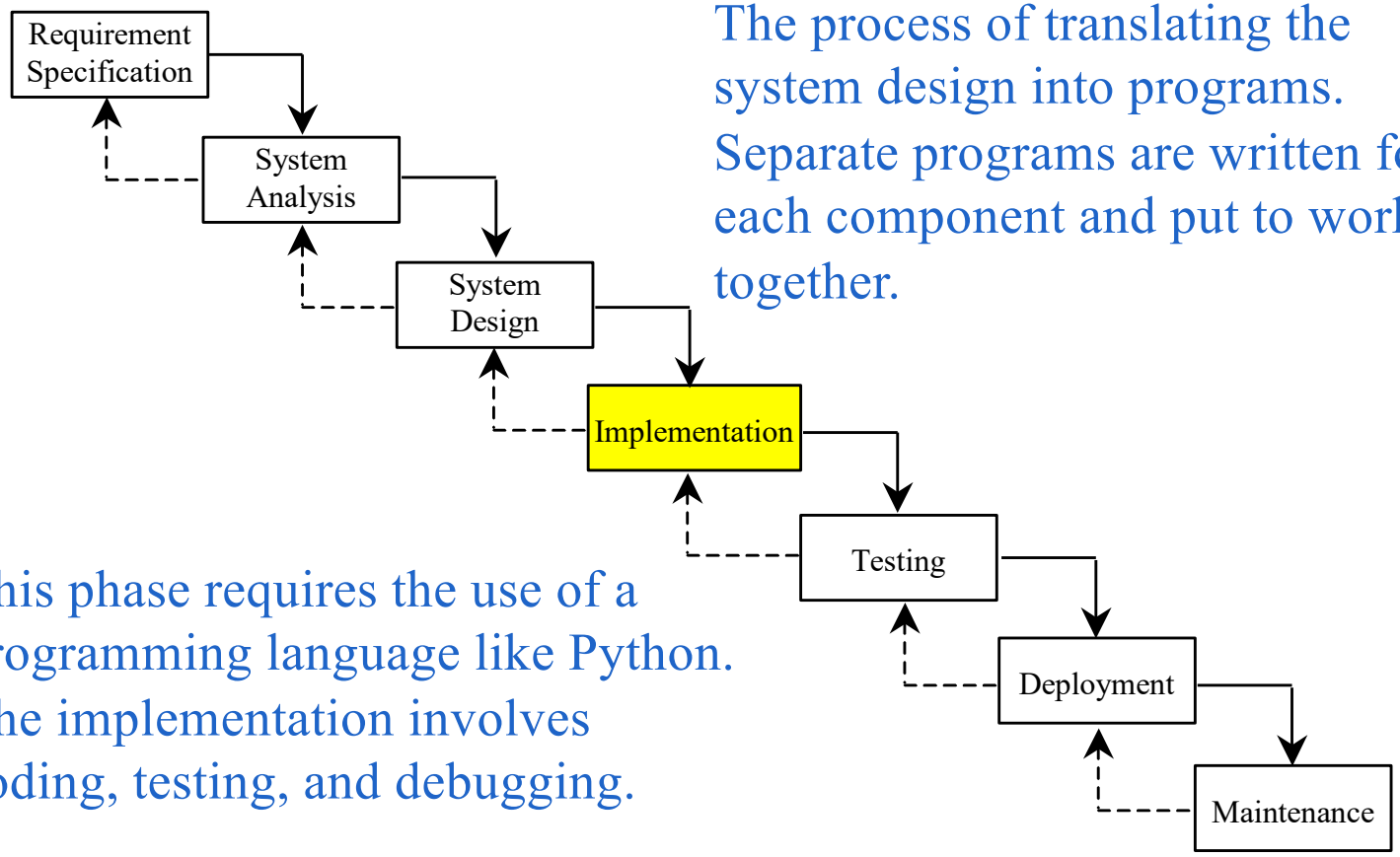
IPO



The essence of system analysis and design is input, process, and output. This is called *IPO*.

UNIVERSITEIT GENT

IMPLEMENTATION

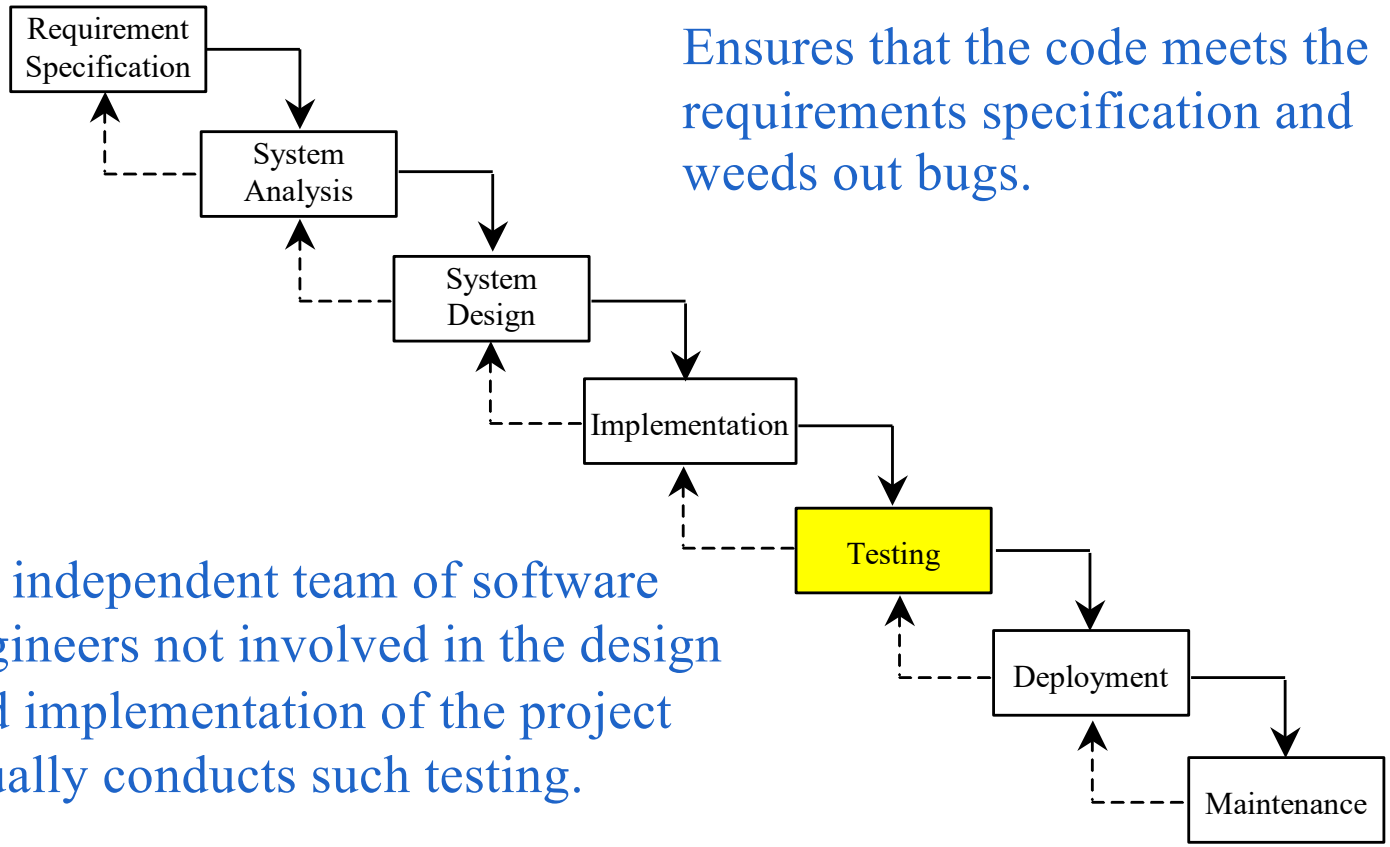


The process of translating the system design into programs. Separate programs are written for each component and put to work together.

This phase requires the use of a programming language like Python. The implementation involves coding, testing, and debugging.

UNIVERSITEIT GENT

TESTING

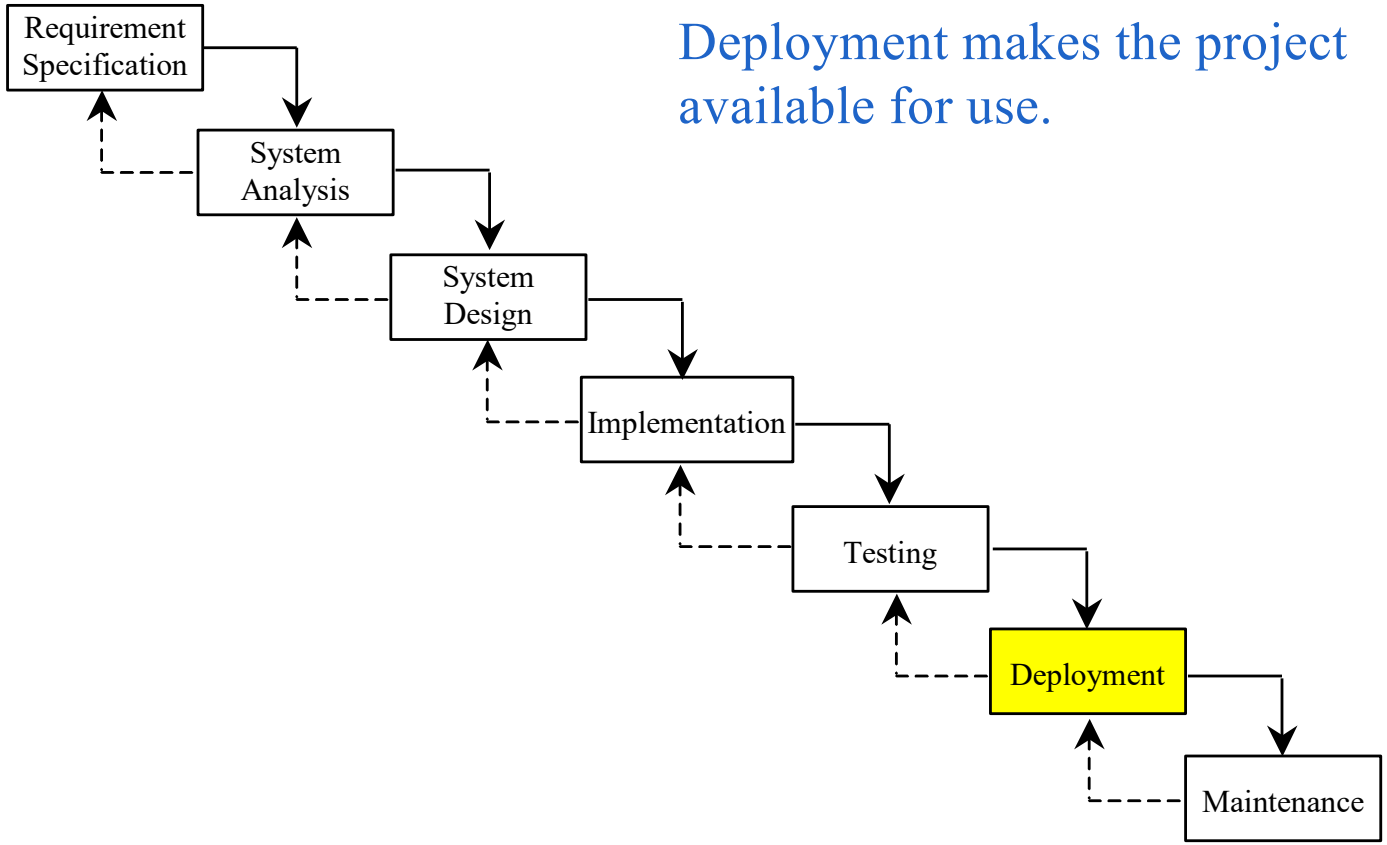


Ensures that the code meets the requirements specification and weeds out bugs.

An independent team of software engineers not involved in the design and implementation of the project usually conducts such testing.

UNIVERSITEIT GENT

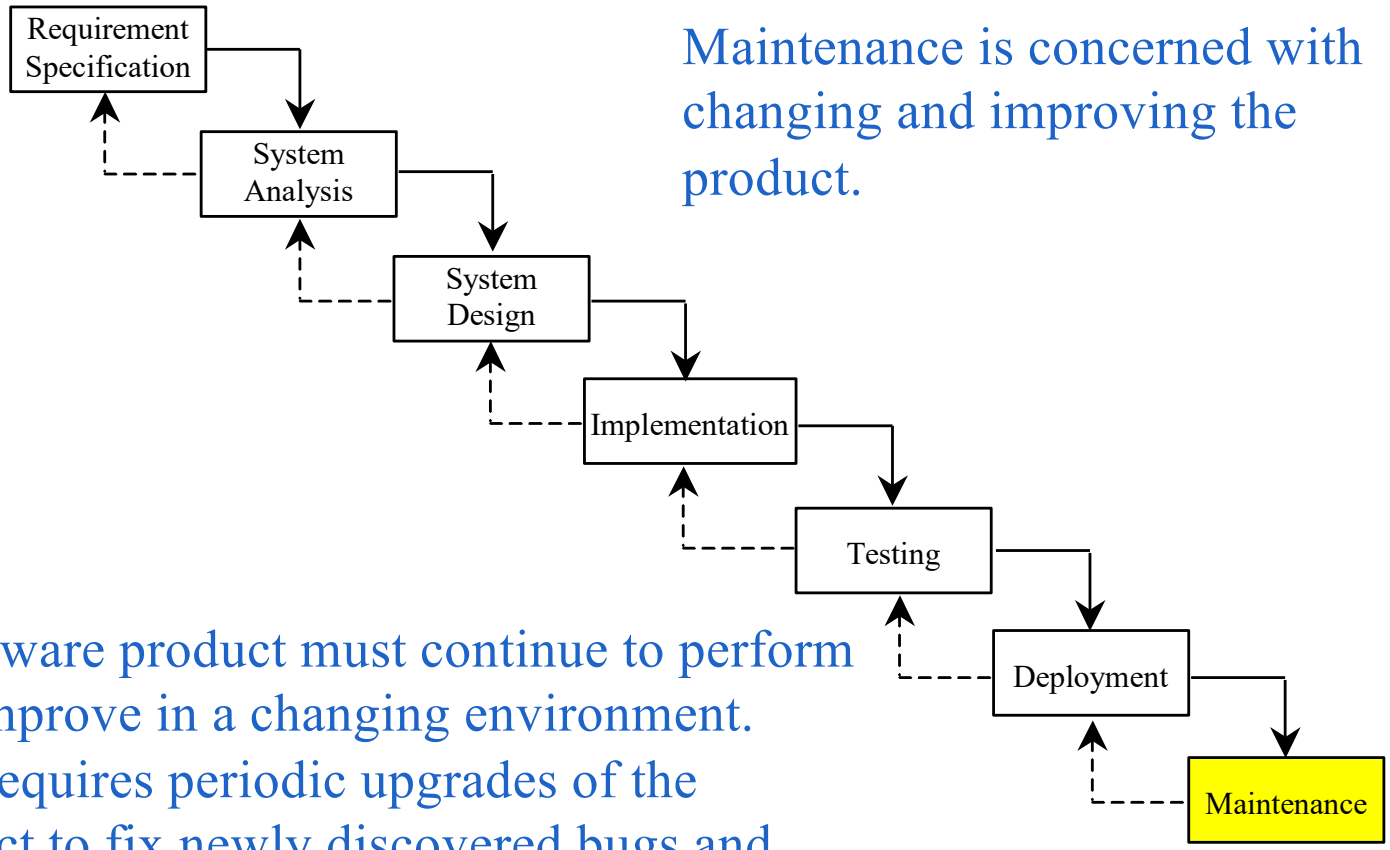
DEPLOYMENT



Deployment makes the project available for use.

UNIVERSITEIT GENT

MAINTENANCE



Maintenance is concerned with changing and improving the product.

A software product must continue to perform and improve in a changing environment. This requires periodic upgrades of the product to fix newly discovered bugs and incorporate changes.

UNIVERSITEIT GENT

PROBLEM: COMPUTING LOAN PAYMENTS

This program lets the user enter the interest rate, number of years, and loan amount, and computes monthly payment and total payment.

$$monthlyPayment = \frac{loanAmount \times monthlyInterestRate}{1 - \frac{1}{(1 + monthlyInterestRate)^{numberOfYears \times 12}}}$$

CASE STUDY: COMPUTING DISTANCES

This program prompts the user to enter two points, computes their distance, and displays the distance.

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

FACULTEIT ECONOMIE
EN BEDRIJFSKUNDE

SELECTIONS

UNIVERSITEIT
GENT

MOTIVATIONS

- If you input a negative value for radius ComputeAreaWithConsoleInput.py the program would print an invalid result. If the radius is negative, you don't want the program to compute the area. How can you deal with this situation?

OBJECTIVES

- To write Boolean expressions using relational operators (§3.2).
- To generate random numbers using the **random.randint(a, b)**, **random.randrange(a, b)**, or **random.random()** functions (§3.3).
- To program with Boolean expressions (**AdditionQuiz**) (§3.3).
- To implement selection control using one-way **if** statements (§3.4).
- To implement selection control using two-way **if-else** statements (§3.5).
- To implement selection control with nested **if** and multi-way **if-elif-else** statements (§3.6).
- To avoid common errors in **if** statements (§3.7).
- To program with selection statements (§§3.8–3.9).
- To combine conditions using logical operators (**and**, **or**, and **not**) (§3.10).
- To use selection statements with combined conditions (**LeapYear**, **Lottery**) (§§3.11–3.12).
- To write expressions that use the conditional expressions (§3.13).
- To understand the rules governing operator precedence and associativity (§3.14).
- To detect the location of an object (§3.15).

BOOLEAN DATA TYPES

- Often in a program you need to compare two values, such as whether i is greater than j. There are six comparison operators (also known as relational operators) that can be used to compare two values. The result of the comparison is a Boolean value: true or false.
- b = (1 > 2)

UNIVERSITEIT GENT

RELATIONAL OPERATORS

Operator	*Name*
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal to

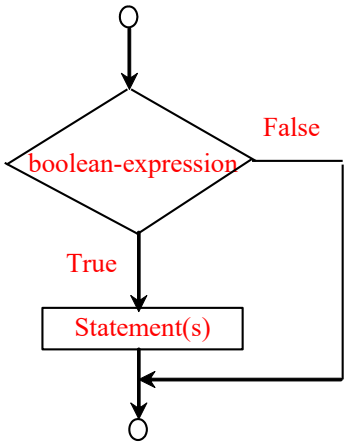
PROBLEM: A SIMPLE MATH LEARNING TOOL

This example creates a program to let a first grader practice additions. The program randomly generates two single-digit integers number1 and number2 and displays a question such as “What is 7 + 9?” to the student. After the student types the answer, the program displays a message to indicate whether the answer is true or false.

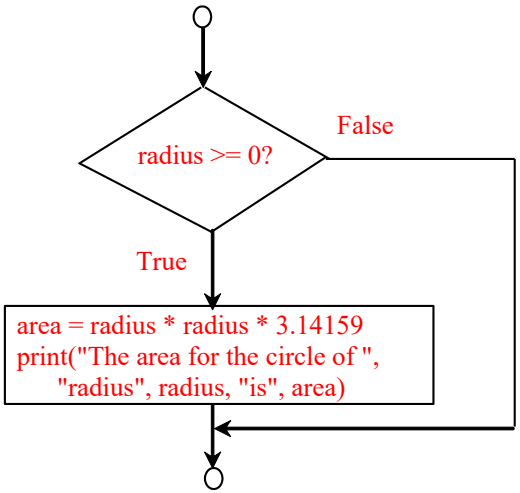
ONE-WAY IF STATEMENTS

```
if boolean-expression:
    statement(s)
```

```
if radius >= 0:
    area = radius * radius * 3.14159
    print("The area for the circle of radius“,
          radius, "is“, area)
```



(a)



(b)

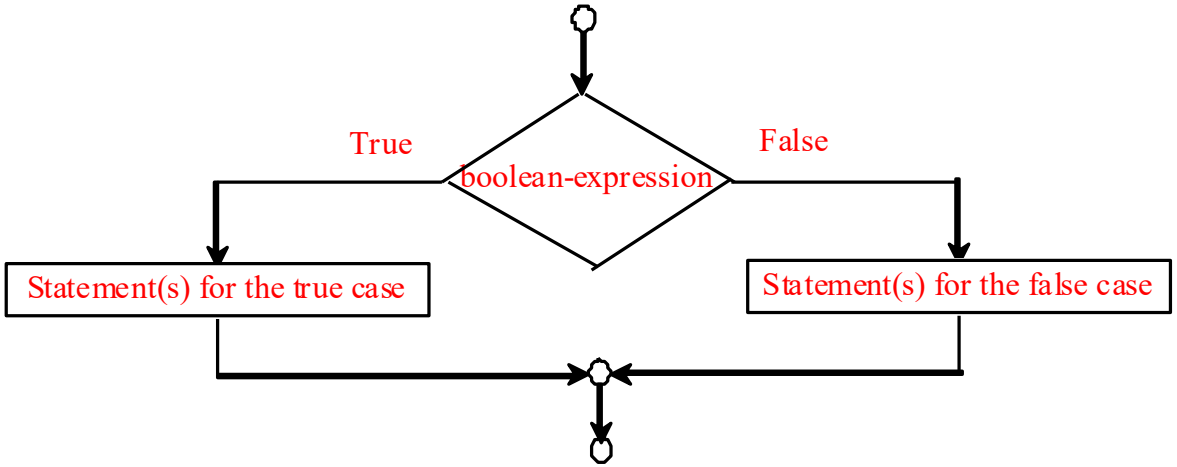
UNIVERSITEIT GENT

SIMPLE IF DEMO

- Write a program that prompts the user to enter an integer. If the number is a multiple of 5, print HiFive. If the number is divisible by 2, print HiEven.

THE TWO-WAY IF STATEMENT

```
if boolean-expression:
    statement(s)-for-the-true-case
else:
    statement(s)-for-the-false-case
```



UNIVERSITEIT GENT

PROBLEM: AN IMPROVED MATH LEARNING TOOL

This example creates a program to teach a first grade child how to learn subtractions. The program randomly generates two single-digit integers number1 and number2 with number1 >= number2 and displays a question such as “What is 9 – 2?” to the student. After the student types the answer, the program displays a message to indicate whether the answer is correct.

MULTIPLE ALTERNATIVE IF STATEMENTS

```
if score >= 90.0:
    grade = 'A'
else:
    if score >= 80.0:
        grade = 'B'
    else:
        if score >= 70.0:
            grade = 'C'
        else:
            if score >= 60.0:
                grade = 'D'
            else:
                grade = 'F'
```

(a)

Equivalent

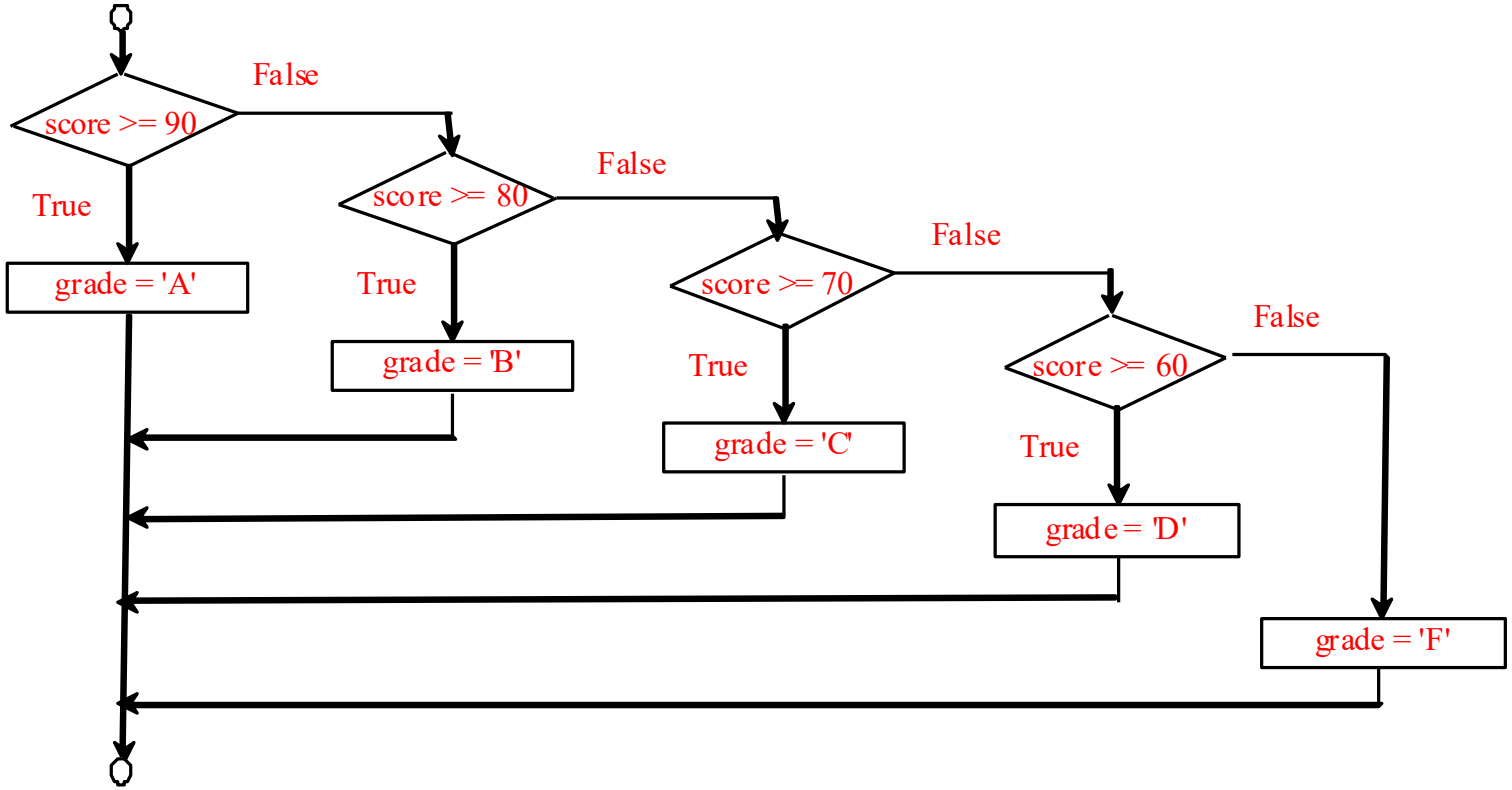
This is better

```
if score >= 90.0:
    grade = 'A'
elif score >= 80.0:
    grade = 'B'
elif score >= 70.0:
    grade = 'C'
elif score >= 60.0:
    grade = 'D'
else:
    grade = 'F'
```

(b)

UNIVERSITEIT GENT

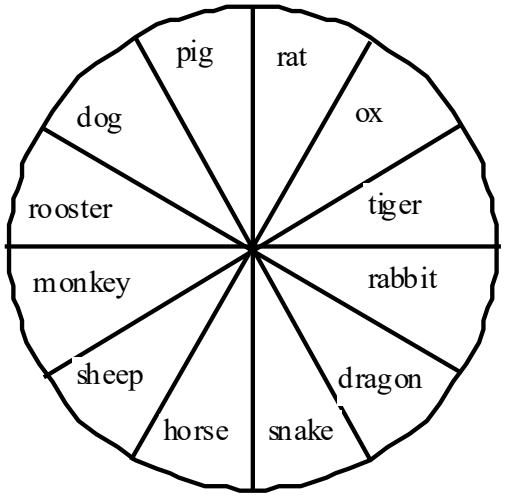
FLOWCHART



UNIVERSITEIT GENT

EXAMPLE

Now let us write a program to find out the Chinese Zodiac sign for a given year. The Chinese Zodiac sign is based on a 12-year cycle, each year being represented by an animal: rat, ox, tiger, rabbit, dragon, snake, horse, sheep, monkey, rooster, dog, and pig, in this cycle.



year % 12 =

0: monkey
1: rooster
2: dog
3: pig
4: rat
5: ox
6: tiger
7: rabbit
8: dragon
9: snake
10: horse
11: sheep

UNIVERSITEIT GENT

COMMON ERRORS

Most common errors in selection statements are caused by incorrect indentation. Consider the following code in (a) and (b).

```
radius = -20

if radius >= 0:
    area = radius * radius * 3.14
print("The area is", area)
```

(a) Wrong

```
radius = -20

if radius >= 0:
    area = radius * radius * 3.14
    print("The area is", area)
```

(b) Correct

UNIVERSITEIT GENT

TIP

```
if number % 2 == 0:
    even = True
else:
    even = False
```

(a)

Equivalent

This is shorter

```
even = number % 2 == 0
```

(b)

UNIVERSITEIT GENT

PROBLEM: BODY MASS INDEX

Body Mass Index (BMI) is a measure of health on weight. It can be calculated by taking your weight in kilograms and dividing by the square of your height in meters. The interpretation of BMI for people 16 years or older is as follows:

BMI	Interpretation
Below 18.5	Underweight
18.5-24.9	Normal
25.0-29.9	Overweight
Above 30.0	Obese

UNIVERSITEIT GENT

PROBLEM: COMPUTING TAXES

The US federal personal income tax is calculated based on the filing status and taxable income. There are four filing statuses: single filers, married filing jointly, married filing separately, and head of household. The tax rates for 2009 are shown below.

Marginal Tax Rate	Single	Married Filing Jointly or Qualified Widow(er)	Married Filing Separately	Head of Household
10%	$0 – $8,350	$0 – $16,700	$0 – $8,350	$0 – $11,950
15%	$8,351– $33,950	$16,701 – $67,900	$8,351 – $33,950	$11,951 – $45,500
25%	$33,951 – $82,250	$67,901 – $137,050	$33,951 – $68,525	$45,501 – $117,450
28%	$82,251 – $171,550	$137,051 – $208,850	$68,525 – $104,425	$117,451 – $190,200
33%	$171,551 – $372,950	$208,851 – $372,950	$104,426 – $186,475	$190,201 - $372,950
35%	$372,951+	$372,951+	$186,476+	$372,951+

PROBLEM: COMPUTING TAXES, CONT.

```
if status == 0:
    # Compute tax for single filers
elif status == 1:
    # Compute tax for married filing jointly
elif status == 2:
    # Compute tax for married filing separately
elif status == 3:
    # Compute tax for head of household
else:
    # Display wrong status
```

LOGICAL OPERATORS

Operator	Description
`not`	`logical negation`
`and`	`logical conjunction`
`or`	`logical disjunction`

TRUTH TABLE FOR LOGICAL OPERATORS

p	not p	Example (assume age = 24, gender = 'F')
True	False	not (age > 18) is False, because (age > 18) is True.
False	True	not (gender == 'M') is True, because (grade == 'M') is False.

p1	p2	p1 and p2	Example (assume age = 24, weight = 140)
False	False	False	
False	True	False	(age > 18) and (weight >= 140) is False, because weight > 140) is False.
True	False	False	
True	True	True	(age > 18) and (weight <= 140) is True, because (age > 18) and (weight <= 140) are both True.

p1	p2	p1 or p2	Example (assume age = 24, weight = 140)
False	False	False	(age > 34) or (weight >= 150) is False, because (age > 34) and (weight >= 150) are both False.
False	True	True	
True	False	True	(age > 14) or (weight < 140) is true, because (age > 14) is True.
True	True	True	

UNIVERSITEIT GENT

OEFENINGEN DODONA

- Write a program that checks whether a number is divisible by 2 and 3, whether a number is divisible by 2 or 3, and whether a number is divisible by 2 or 3 but not both
- Write a program that first prompts the user to enter a year as an int value and checks if it is a leap year.

OEFENINGEN DODONA

- Write a program that randomly generates a lottery between 0 and 99, prompts the user to enter a number in the same range, and determines whether the user wins according to the following rule:
  - If the user input matches the lottery in exact order, the award is 10.000 €.
  - If the user input matches the lottery, the award is 3.000 €.
  - If one digit in the user input matches a digit in the lottery, the award is 1.000 €.

CONDITIONAL EXPRESSIONS

```
if x > 0:
    y = 1
else:
    y = -1
```

is equivalent to

```
y = 1 if x > 0 else -1
```

```
expression1 if boolean-expression else expression2
```

UNIVERSITEIT GENT

CONDITIONAL OPERATOR

```
if number % 2 == 0:
    print(number, "is even")
else:
    print(number, "is odd")
```

```
print(number, "is even" if (number % 2 ==
0) else "is odd")
```

MATCH-CASE STATEMENTS (PYTHON 3.10)

```
match expression:
  case value1: statement(s) 1
  case value2: statement(s) 2
  ….
  case valueN: statement(s) N
  case _:  statements for default case
```

OPERATOR PRECEDENCE

- +, -
- **
- not
- *, /, //, %
- +, -
- <, <=, >, >=
- ==, !=
- and
- or
- =, +=, -=, *=, /=, //=, %= (Assignment operator)

UNIVERSITEIT
GENT

OPERATOR PRECEDENCE AND ASSOCIATIVITY

- The expression in the parentheses is evaluated first. (Parentheses can be nested, in which case the expression in the inner parentheses is executed first.) When evaluating an expression without parentheses, the operators are applied according to the precedence rule and the associativity rule.

- If operators with the same precedence are next to each other, their associativity determines the order of evaluation. All binary operators except assignment operators are left-associative.

OPERATOR ASSOCIATIVITY

- When two operators with the same precedence are evaluated, the associativity of the operators determines the order of evaluation. All binary operators except assignment operators are left-associative.
  a – b + c – d is equivalent to ((a – b) + c) – d
- Assignment operators are right-associative. Therefore, the expression
  a = b = c = 5 is equivalent to a = (b = (c = 5))

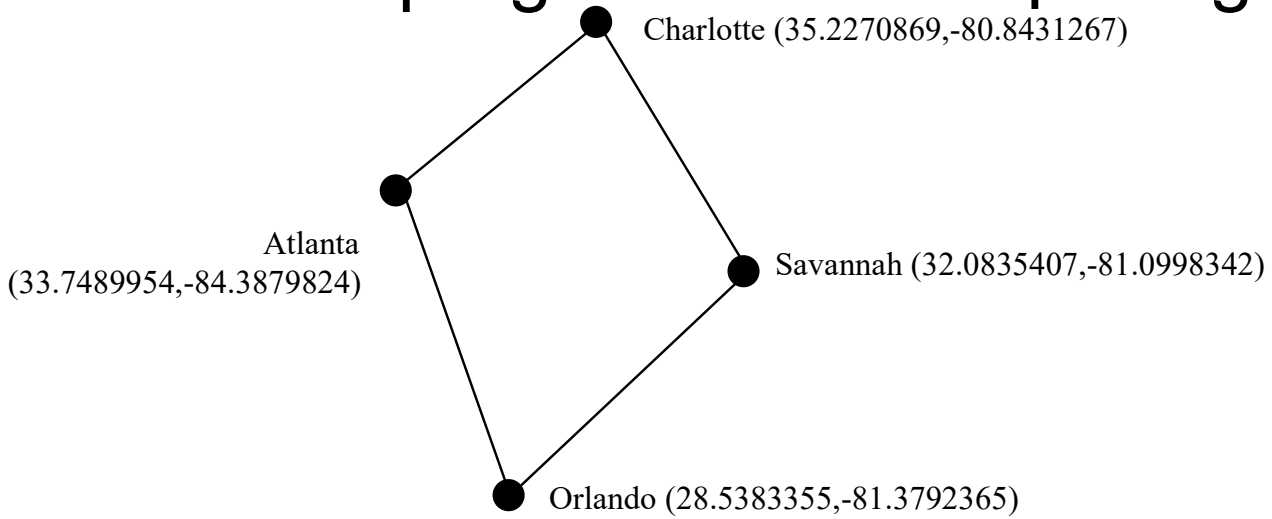
FACULTEIT ECONOMIE
EN BEDRIJFSKUNDE

MATHEMATICAL FUNCTIONS, STRINGS, AND OBJECTS

UNIVERSITEIT
GENT

MOTIVATIONS

Suppose you need to estimate the area enclosed by four cities, given the GPS locations (latitude and longitude) of these cities, as shown in the following diagram. How would you write a program to solve this problem? You will be able to write such a program after completing this chapter.



UNIVERSITEIT GENT

OBJECTIVES

- To solve mathematics problems by using the functions in the math module (§4.2).
- To represent and process strings and characters (§§4.3-4.4).
- To encode characters using ASCII and Unicode (§§4.3.1-4.3.2).
- To use the ord function to obtain a numerical code for a character and the chr function to convert a numerical code to a character (§4.3.3).
- To represent special characters using the escape sequence (§4.3.4).
- To invoke the print function with the end argument (§4.3.5).
- To convert numbers to a string using the str function (§4.3.6).
- To use the + operator to concatenate strings (§4.3.7).
- To read strings from the keyboard (§4.3.8).
- To solve the lottery problem using strings (§4.4).
- To introduce objects and methods (§4.5).
- To introduce the methods in the str class (§4.6).
- To program using characters and strings (GuessBirthday) (<LINK>§4.7.1</LINK>).
- To convert a hexadecimal character to a decimal value (HexDigit2Dec) (<LINK>§4.7.2</LINK>).
- To format numbers and strings using the format function (§4.8).

PYTHON BUILT-IN FUNCTIONS

Function	Description	Example
abs(x)	Returns the absolute value for x.	abs(-2) is 2
max(x1, x2, ...)	Returns the largest among x1, x2, ...	max(1, 5, 2) is 5
min(x1, x2, ...)	Returns the smallest among x1, x2, ...	min(1, 5, 2) is 1
pow(a, b)	Returns a^b. Same as a ** b.	pow(2, 3) is 8
round(x)	Returns an integer nearest to x. If x is equally close to two integers, the even one is returned.	round(5.4) is 5 round(5.5) is 6 round(4.5) is 4
round(x, n)	Returns the float value rounded to n digits after the decimal point.	round(5.466, 2) is 5.47 round(5.463, 2) is 5.46

UNIVERSITEIT GENT

BUILT-IN FUNCTIONS

```
>>> max(2, 3, 4) # Returns a maximum number
4
>>> min(2, 3, 4) # Returns a minimum number
2
>>> round(4.51) # Rounds to its nearest integer
4
>>> round(4.4) # Rounds to its nearest integer
4
>>> abs(-3) # Returns the absolute value
3
>>> pow(2, 3) # Same as 2 ** 3
8
```

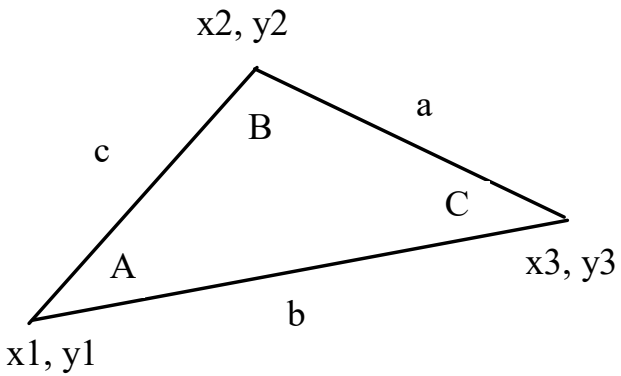
THE MATH FUNCTIONS

Function	Description	Example
fabs(x)	Returns the absolute value of the argument.	fabs(-2) is 2
ceil(x)	Rounds x up to its nearest integer and returns this integer.	ceil(2.1) is 3 ceil(-2.1) is -2
floor(x)	Rounds x down to its nearest integer and returns this integer.	floor(2.1) is 2 floor(-2.1) is -3
exp(x)	Returns the exponential function of x (e ** x).	exp(1) is 2.71828
log(x)	Returns the natural logarithm of x.	log(2.71828) is 1.0
log(x, base)	Returns the logarithm of x for the specified base.	log10(10, 10) is 1
sqrt(x)	Returns the square root of x.	sqrt(4.0) is 2
sin(x)	Returns the sine of x. x represents an angle in radians.	sin(3.14159 / 2) is 1 sin(3.14159) is 0
asin(x)	Returns the angle in radians for the inverse of sine.	asin(1.0) is 1.57 asin(0.5) is 0.523599
cos(x)	Returns the cosine of x. x represents an angle in radians.	cos(3.14159 / 2) is 0 cos(3.14159) is -1
acos(x)	Returns the angle in radians for the inverse of cosine.	acos(1.0) is 0 acos(0.5) is 1.0472
tan(x)	Returns the tangent of x. x represents an angle in radians.	tan(3.14159 / 4) is 1 tan(0.0) is 0
fmod(x, y)	Returns the remainder of x/y as double.	fmod(2.4, 1.3) is 1.1
degrees(x)	Converts angle x from radians to degrees	degrees(1.57) is 90
radians(x)	Converts angle x from degrees to radians	radians(90) is 1.57

UNIVERSITEIT GENT

PROBLEM: COMPUTE ANGLES

Given three points of a triangle, you can compute the angles using the following formula:




```
A = acos((a * a - b * b - c * c) / (-2 * b * c))
B = acos((b * b - a * a - c * c) / (-2 * a * c))
C = acos((c * c - b * b - a * a) / (-2 * a * b))
```

UNIVERSITEIT GENT

STRINGS AND CHARACTERS

A string is a sequence of characters. *String* literals can be enclosed in matching *single quotes* (') or *double quotes* ("). Python does not have a data type for characters. A single-character string represents a character.

```
Letter = 'A' # Same as letter = "A"
numChar = '4' # Same as numChar = "4"
message = "Good morning" # Same as message = 'Good morning'
```

NOTE

For consistency, this book uses double quotes for a string with more than one character and single quotes for a string with a single character or an empty string. This convention is consistent with other programming languages. So, it will be easy to convert a Python program to a program written in other languages.

UNICODE AND ASCII CODE

Python characters use *Unicode*, a 16-bit encoding scheme. Unicode is an encoding scheme for representing international characters. ASCII is a small subset of Unicode.

TABLE B.1 ASCII Character Set in the Decimal Index

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	nl	vt	ff	cr	so	si	dle	dcl	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	″	#	$	%	&	’
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	‘	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	del		

UNIVERSITEIT GENT

FUNCTIONS ORD AND CHR

```
>>> ch = 'a'
>>> ord(ch)
>>> 97
>>> chr(98)
>>> 'b'
```

ESCAPE SEQUENCES FOR SPECIAL CHARACTERS

Description	*Escape Sequence*	*Unicode*
Backspace	`\b`	`\u0008`
Tab	`\t`	`\u0009`
Linefeed	`\n`	`\u000A`
Carriage return	`\r`	`\u000D`
Backslash	`\\`	`\u005C`
Single Quote	`\'`	`\u0027`
Double Quote	`\"`	`\u0022`

PRINTING WITHOUT THE NEWLINE

```
print(item, end = 'anyendingstring')
```

```
print("AAA", end = ' ')
print("BBB", end = '')
print("CCC", end = '***')
print("DDD", end = '***')
```

```
print(item1, item2, sep = '\n')
```

UNIVERSITEIT
GENT

THE STR FUNCTION

The str function can be used to convert a number into a string. For example,

```
>>> s = str(4.4) # Convert a float to string
>>> s
'4.4'
>>> s = str(3) # Convert an integer to string
>>> s
'3'
>>>
```

THE STRING CONCATENATION OPERATOR

You can use the + operator add two numbers. The + operator can also be used to concatenate (combine) two strings. Here are some examples:

```
>>> message = "Welcome " + "to " + "Python"
>>> message
'Weclome to Python'
>>> chapterNo = 2
>>> s = "Chapter " + str(chapterNo)
>>> s
'Chapter 2'
>>>
```

READING STRINGS FROM THE CONSOLE

To read a string from the console, use the input function. For example, the following code reads three strings from the keyboard:

```
s1 = input("Enter a string: ")
s2 = input("Enter a string: ")
s3 = input("Enter a string: ")
print("s1 is " + s1)
print("s2 is " + s2)
print("s3 is " + s3)
```

IN AND NOT IN OPERATORS

Can be used to test whether a string is part of another string:

```
>>> s1 = ”Welcome”
>>> ”come” in s1
True
>>> ”come” not in s1
False
```

COMPARING STRINGS

- Use ==,!=, >, >=, < and <=
- Python compare strings by comparing their corresponding characters and by evaluating the corresponding characters' numeric codes.

```
>>>"green" == "glow"
False
>>> "green" != "glow"
True
```

UNIVERSITEIT GENT

BUILT-IN FUNCTION LEN, MAX, MIN

– The len function returns number of characters in a string

```
>>> s = ”Welcome”
>>> len(s)
7
>>> max(s)
‘o’
>>> min(s)
‘W’
```

UNIVERSITEIT GENT

INDEX OPERATOR [ ]

- A string is a sequence of characters. A character in a String can be accessed through the index operator using the syntax: s[index]
- Python also allows negative inedexes to reference positions relative to the end of the string

```
>>> s = ”Welcome”
>>> s[0]
’W’
>>> s[1]
’e’
>>> s[-1] # ➔ s[-1 + len(s)]
’e’
```

SLICING OPERATOR [START : END]

The slicing operator returns a slice of the string using the syntax s[start : end].
The slice us a substring from index start to index end – 1

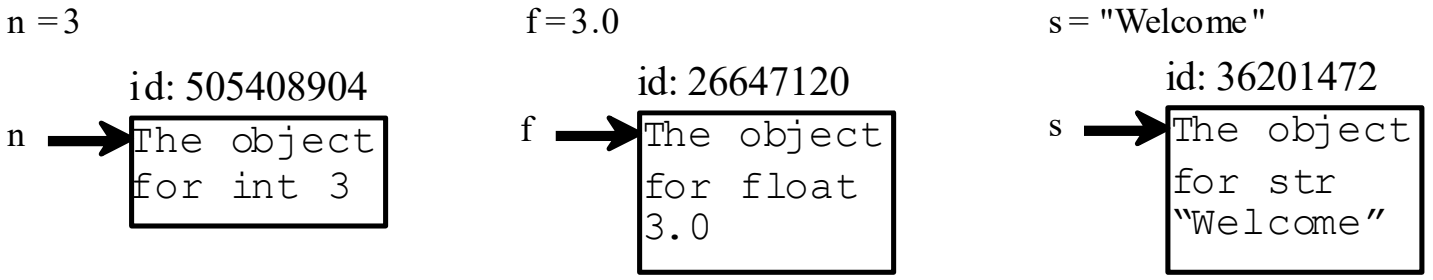
```
>>> s = ”Welcome”
>>> s[1:4]
”elc”
>>> s[:6]
”Welcom”
>>> s[4:]
”ome”
>>> s[1:-1]
”elcome”
```

INTRODUCTION TO OBJECTS AND METHODS

- In Python, all data—including numbers and strings—are actually objects.
- An object is an entity. Each object has an id and a type. Objects of the same kind have the same type. You can use the id function and type function to get these information for an object.
- The **id** and **type** functions are rarely used in programming, but they are good pedagogical tools for understanding objects.

OBJECT VS. OBJECT REFERENCE VARIABLE

- For n = 3, we say n is an integer variable that holds value 4. Strictly speaking, n is a variable that references an int object for value 4. For simplicity, it is fine to say n is an int variable with value 4.



UNIVERSITEIT GENT

METHODS

You can perform operations on an object. The operations are defined using functions. The functions for the objects are called *methods* in Python. Methods can only be invoked from a specific object. For example, the string type has the methods such as lower() and upper(), which returns a new string in lowercase and uppercase.

TESTING STRINGS

Method	Description
isalnum()	Returns True if all characters in this string are alphanumeric and there is at least one character.
isalpha()	Returns True if all characters in this string are alphabetic and there is at least one character.
isdigit()	Returns True if this string contains only number characters.
isidentifier()	Returns True if this string is a Python identifier.
islower()	Returns True if all characters in this string are lowercase letters and there is at least one character.
isupper()	Returns True if all characters in this string are uppercase letters and there is at least one character.
isspace()	Returns True if this string contains only whitespace characters.

SEARCHING FOR SUBSTRINGS

Method	Description
endswith(s1)	Returns True if the string ends with the substring s1.
startswith(s1)	Returns True if the string starts with the substring s1.
find(s1)	Returns the lowest index where s1 starts in this string, or -1 if s1 is not found in this string.
rfind(s1)	Returns the highest index where s1 starts in this string, or -1 if s1 is not found in this string.
count(subtring)	Returns the number of non-overlapping occurrences of this substring.

CONVERTING STRINGS

Method	Description
capitalize()	Returns a copy of this string with only the first character capitalized.
lower()	Returns a copy of this string with all letters converted to lowercase.
upper()	Returns a copy of this string with all letters converted to uppercase.
title()	Returns a copy of this string with the first letter capitalized in each word.
swapcase()	Returns a copy of this string in which lowercase letters are converted to upper and uppercase to lowercase.
replace(old, new)	Returns a new string that replaces all the occurrence of the old string with a n string.
replace(old, new, n)	Returns a new string that replaces up to n number of the occurrence of the old with a new string.

STRIPING WHITESPACE CHARACTERS

Method	Description
lstrip()	Returns a string with the leading whitespace characters removed.
rstrip()	Returns a string with the trailing whitespace characters removed.
strip()	Returns a string with the starting and trailing whitespace characters removed.

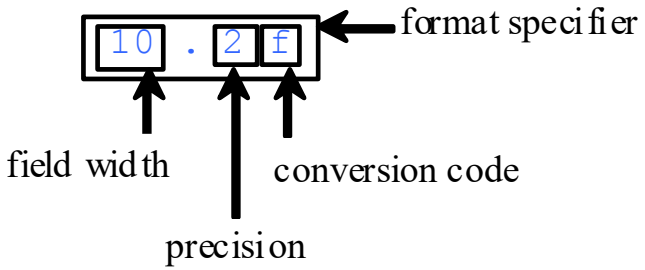
FORMATTING NUMBERS AND STRINGS

- Often it is desirable to display numbers in certain format. For example, the following code computes the interest, given the amount and the annual interest rate.
- The format function formats a number or a string and returns a string.

format(item, format-specifier)

FORMATTING FLOATING-POINT NUMBERS

```
print(format(57.467657, '10.2f'))
print(format(12345678.923, '10.2f'))
print(format(57.4, '10.2f'))
print(format(57, '10.2f'))
```



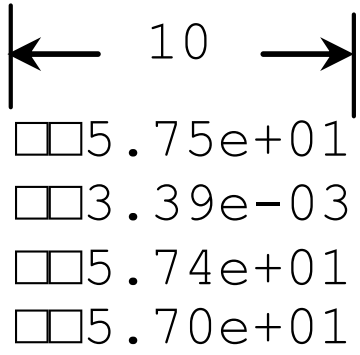

```
□□□□57.47
12345678.92
□□□□57.40
□□□□57.00
```

UNIVERSITEIT GENT

FORMATTING IN SCIENTIFIC NOTATION

If you change the conversion code from f to e, the number will be formatted in scientific notation. For example,

```
print(format(57.467657, '10.2e'))
print(format(0.0033923, '10.2e'))
print(format(57.4, '10.2e'))
print(format(57, '10.2e'))
```

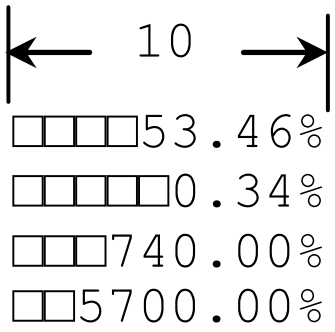


UNIVERSITEIT GENT

FORMATTING AS A PERCENTAGE

You can use the conversion code % to format numbers as a percentage. For example,

```
print(format(0.53457, '10.2%'))
print(format(0.0033923, '10.2%'))
print(format(7.4, '10.2%'))
print(format(57, '10.2%'))
```

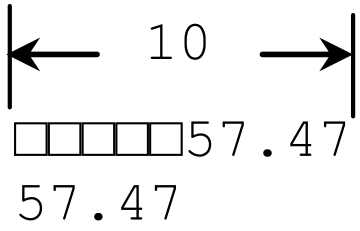


UNIVERSITEIT GENT

JUSTIFYING FORMAT

By default, the format is right justified. You can put the symbol < in the format specifier to specify that the item is a left justified in the resulting format within the specified width. For example,

```
print(format(57.467657, '10.2f'))
print(format(57.467657, '<10.2f'))
```

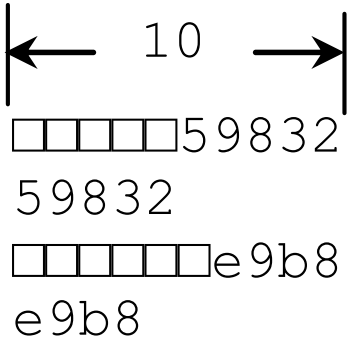


UNIVERSITEIT GENT

FORMATTING INTEGERS

You can use the conversion code d, x, o, and b to format an integer in decimal, hexadecimal, octal, or binary. You can specify a width for the conversion. For example,

```
print(format(59832, '10d'))
print(format(59832, '<10d'))
print(format(59832, '10x'))
print(format(59832, '<10x'))
```

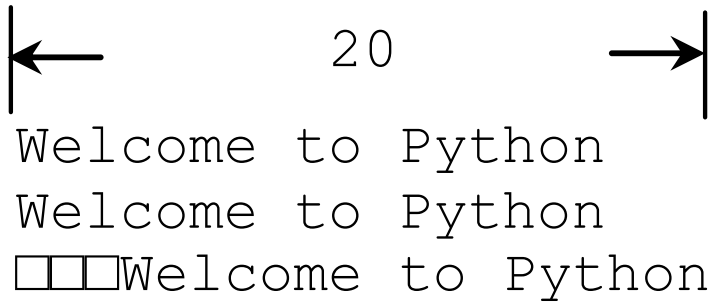


UNIVERSITEIT GENT

FORMATTING STRINGS

You can use the conversion code s to format a string with a specified width. For example,

```
print(format("Welcome to Python", '20s'))
print(format("Welcome to Python", '<20s'))
print(format("Welcome to Python", '>20s'))
```



UNIVERSITEIT GENT

F-STRINGS AND FORMAT

F-strings can replace the format function:

format(item,”format-specifier”) ➔ f”{item:format-specifier}

```
>>> weight = 140
>>> height = 73
>>> f”Weight is {weight} and height is {height}.”
```

F-STRINGS

A string that begins with f or F. The may contains expressions inside brackets. The expressions are evaluated at runtime and formatted to strings

```
>>> weight = 140
>>> height = 73
>>> f”Weight is {weight} and height is {height}.”
```

CHAPTER 5 LOOPS

UNIVERSITEIT
GENT

MOTIVATIONS

Suppose that you need to print a string (e.g., "Programming is fun!") a hundred times. It would be tedious to have to write the following statement a hundred times:

```
print("Programming is fun!");
```

So, how do you solve this problem?

OPENING PROBLEM

Problem:

100 times

```
print("Programming is fun!");
print("Programming is fun!");
print("Programming is fun!");
print("Programming is fun!");
print("Programming is fun!");
print("Programming is fun!");

...

...

...
print("Programming is fun!");
print("Programming is fun!");
print("Programming is fun!");
```

UNIVERSITEIT GENT

INTRODUCING WHILE LOOPS

```
count = 0
while count < 100:
    print("Programming is fun!")
    count = count + 1
```

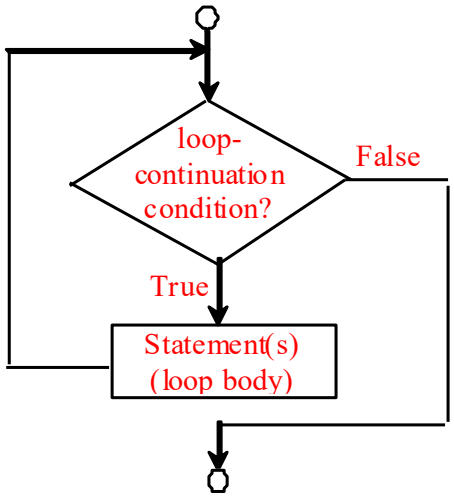
OBJECTIVES

- To write programs for executing statements repeatedly by using a **while** loop (§5.2).
- To develop loops following the loop design strategy (§§5.2.1-5.2.3).
- To control a loop with the user's confirmation (§5.2.4).
- To control a loop with a sentinel value (§5.2.5).
- To use **for** loops to implement counter-controlled loops (§5.3).
- To write nested loops (§5.4).
- To learn the techniques for minimizing numerical errors (§5.5).
- To learn loops from a variety of examples (**GCD**, **FutureTuition**, **PrimeNumber**) (§§5.6, 5.8).
- To implement program control with **break** and **continue** (§5.7).

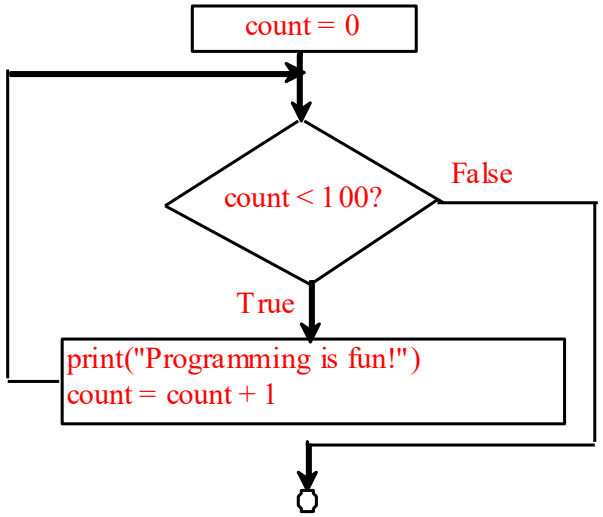
WHILE LOOP FLOW CHART

```
while loop-continuation-
condition:
    # Loop body
    Statement(s)
```

```
count = 0
while count < 100:
    print("Programming is fun!")
    count = count + 1
```



(a)



(b)

PROBLEM: AN ADVANCED MATH LEARNING TOOL

Recall SubtractionQuiz.py, gives a program that prompts the user to enter an answer for a question on subtraction. Using a loop, you can now rewrite the program to let the user enter a new answer until it is correct.

PROBLEM: GUESSING NUMBERS

Write a program that randomly generates an integer between 0 and 100, inclusive. The program prompts the user to enter a number continuously until the number matches the randomly generated number. For each user input, the program tells the user whether the input is too low or too high, so the user can choose the next input intelligently.

CASE STUDY: MULTIPLE SUBTRACTION QUIZ

The Math subtraction learning tool program generates just one question for each run. You can use a loop to generate questions repeatedly. This example gives a program that generates five questions and reports the number of the correct answers after a student answers all five questions.

ENDING A LOOP WITH A SENTINEL VALUE

- Often the number of times a loop is executed is not predetermined. You may use an input value to signify the end of the loop. Such a value is known as a sentinel value.
- Write a program that reads and calculates the sum of an unspecified number of integers. The input 0 signifies the end of the input.

CAUTION

Don’t use floating-point values for equality checking in a loop control. Since floating-point values are approximations for some values, using them could result in imprecise counter values and inaccurate results. Consider the following code for computing 1 + 0.9 + 0.8 + ... + 0.1:

```
item = 1
sum = 0
while item != 0: # No guarantee item will be 0
    sum += item
    item -= 0.1
print(sum)
```

Variable item starts with 1 and is reduced by 0.1 every time the loop body is executed. The loop should terminate when item becomes 0. However, there is no guarantee that item will be exactly 0, because the floating-point arithmetic is approximated. This loop seems OK on the surface, but it is actually an infinite loop.

FOR LOOPS

```
i = initialValue  # Initialize loop-control variable
while i < endValue:
    # Loop body
    ...
    i++ # Adjust loop-control variable

for i in range(initialValue, endValue):
    # Loop body
```

UNIVERSITEIT
GENT

RANGE

- range(A,B)
- range(A)
- range(A,B, step)

```
>>> for v in range(4, 8):
...     print(v)
...
4
5
6
7
>>>
```

UNIVERSITEIT GENT

NESTED LOOPS

Problem: Write a program that uses nested for loops to print a multiplication table.

```
                Multiplication Table
      1   2   3   4   5   6   7   8   9
---------------------------------------
1 |   1   2   3   4   5   6   7   8   9
2 |   2   4   6   8  10  12  14  16  18
3 |   3   6   9  12  15  18  21  24  27
4 |   4   8  12  16  20  24  28  32  36
5 |   5  10  15  20  25  30  35  40  45
6 |   6  12  18  24  30  36  42  48  54
7 |   7  14  21  28  35  42  49  56  63
8 |   8  16  24  32  40  48  56  64  72
9 |   9  18  27  36  45  54  63  72  81
```

UNIVERSITEIT GENT

PROBLEM:
FINDING THE GREATEST COMMON DIVISOR

Problem: Write a program that prompts the user to enter two positive integers and finds their greatest common divisor.

Solution: Suppose you enter two integers 4 and 2, their greatest common divisor is 2. Suppose you enter two integers 16 and 24, their greatest common divisor is 8. So, how do you find the greatest common divisor? Let the two input integers be n1 and n2. You know number 1 is a common divisor, but it may not be the greatest common divisor. So you can check whether k (for k = 2, 3, 4, and so on) is a common divisor for n1 and n2, until k is greater than n1 or n2.

PROBLEM: PREDICTING THE FUTURE TUITION

Problem: Suppose that the tuition for a university is €1.000 this year and tuition increases 7% every year. In how many years will the tuition be doubled?

DODONA: CONVERTING DECIMALS TO HEXADECIMALS

Hexadecimals are often used in computer systems programming (see Appendix F for an introduction to number systems). How do you convert a decimal number to a hexadecimal number? To convert a decimal number *d* to a hexadecimal number is to find the hexadecimal digits h_n, h_{n-1}, h_{n-2}, ... , h_2, h_1, and h_0 such that

$$d = h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + ... + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0$$

These hexadecimal digits can be found by successively dividing *d* by 16 until the quotient is 0. The remainders are h_0, h_1, h_2, ... , h_{n-2}, h_{n-1}, and h_n.

USING BREAK AND CONTINUE

- Examples for using the `break` and `continue` keywords:

Break out of the loop

```
sum = 0
number = 0

while number < 20:
    number += 1
    sum += number
    if sum >= 100:
        break

print("The number is ", number)
print("The sum is ", sum)
```

Jump to the end of the iteration

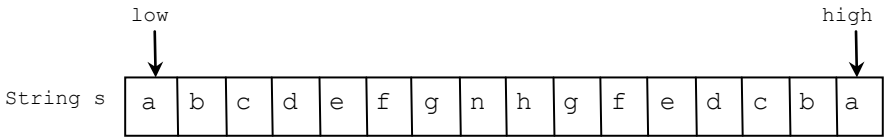
```
sum = 0
number = 0

while number < 20:
    number += 1
    if number == 10 or number == 11:
        continue
    sum += number

print("The sum is ", sum)
```

UNIVERSITEIT GENT

DODONA: CHECKING PALINDROME

A string is a palindrome if it reads the same forward and backward. The words “mom,” “dad,” and “noon,” for instance, are all palindromes.

The problem is to write a program that prompts the user to enter a string and reports whether the string is a palindrome. One solution is to check whether the first character in the string is the same as the last character. If so, check whether the second character is the same as the second-to-last character. This process continues until a mismatch is found or all the characters in the string are checked, except for the middle character if the string has an odd number of characters.



UNIVERSITEIT GENT

DODONA: DISPLAYING PRIME NUMBERS

Problem: Write a program that displays the first 50 prime numbers in five lines, each of which contains 10 numbers. An integer greater than 1 is *prime* if its only positive divisor is 1 or itself. For example, 2, 3, 5, and 7 are prime numbers, but 4, 6, 8, and 9 are not.

Solution: The problem can be broken into the following tasks:

- For number = 2, 3, 4, 5, 6, ..., test whether the number is prime.
- Determine whether a given number is prime.
- Count the prime numbers.
- Print each prime number, and print 10 numbers per line.

FACULTEIT ECONOMIE
EN BEDRIJFSKUNDE

CHAPTER 6 FUNCTIONS

UNIVERSITEIT
GENT

OPENING PROBLEM

Find the sum of integers from 1 to 10, from 20 to 37, and from 35 to 49, respectively.

```
sum = 0
for i in range(1, 11):
    sum += i
print("Sum from 1 to 10 is", sum)

sum = 0
for i in range(20, 38):
    sum += i
print("Sum from 20 to 37 is", sum)

sum = 0
for i in range(35, 50):
    sum += i
print("Sum from 35 to 49 is", sum)
```

PROBLEM

```
sum = 0
for i in range(1, 11):
    sum += i
print("Sum from 1 to 10 is", sum)
```

```
sum = 0
for i in range(20, 38):
    sum += i
print("Sum from 20 to 37 is", sum)
```

```
sum = 0
for i in range(35, 50):
    sum += i
print("Sum from 35 to 49 is", sum)
```

SOLUTION

```
def sum(i1, i2):
    result = 0
    for i in range(i1, i2 + 1):
        result += i
    return result

def main():
    print("Sum from 1 to 10 is", sum(1, 10))
    print("Sum from 20 to 37 is", sum(20, 37))
    print("Sum from 35 to 49 is", sum(35, 49))

main() # Call the main function
```

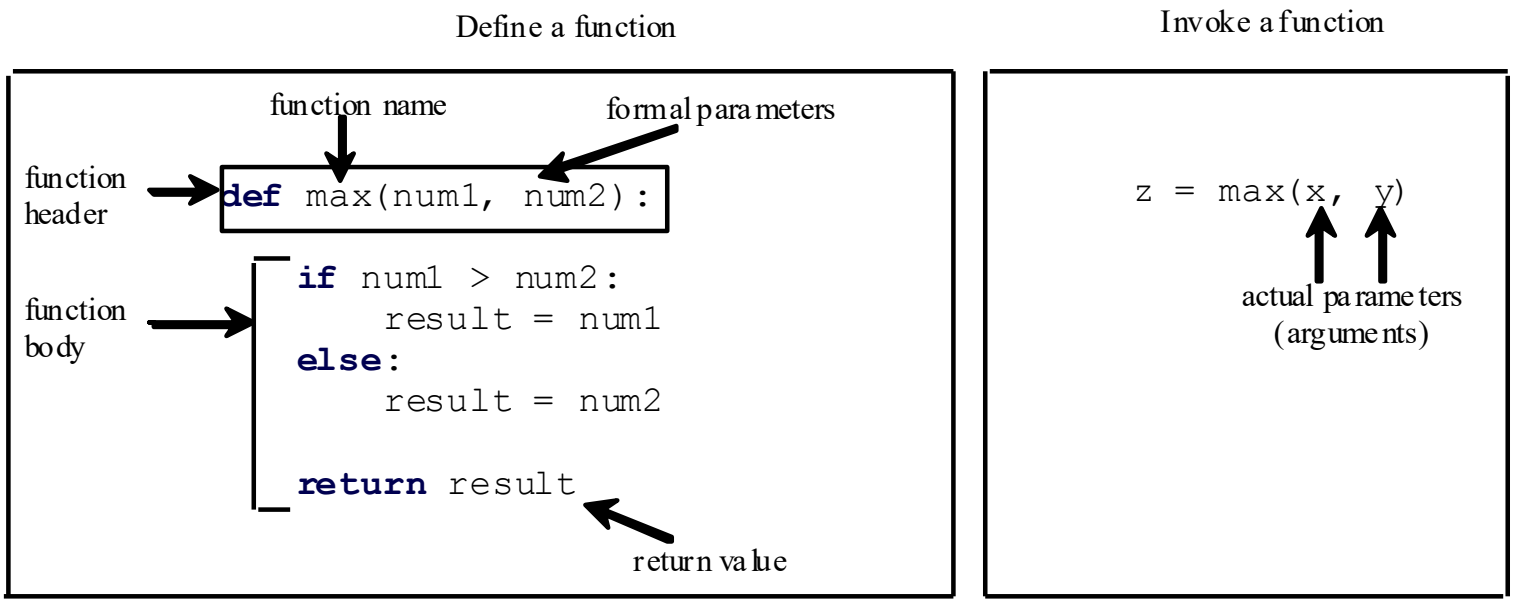
UNIVERSITEIT GENT

OBJECTIVES

- To define functions (§6.2).
- To invoke value-returning functions (§6.3).
- To invoke functions that do not return a value (§6.4).
- To pass arguments by values (§6.5).
- To develop reusable code that is modular, easy to read, easy to debug, and easy to maintain (§6.7).
- To create modules for reusing functions (§§6.7-6.8).
- To determine the scope of variables (§6.9).
- To define functions with default arguments (§6.10).
- To return multiple values from a function (§6.11).
- To apply the concept of function abstraction in software development (§6.12).
- To design and implement functions using stepwise refinement (§6.13).
- To simplify drawing programs using functions (§6.14).

DEFINING FUNCTIONS

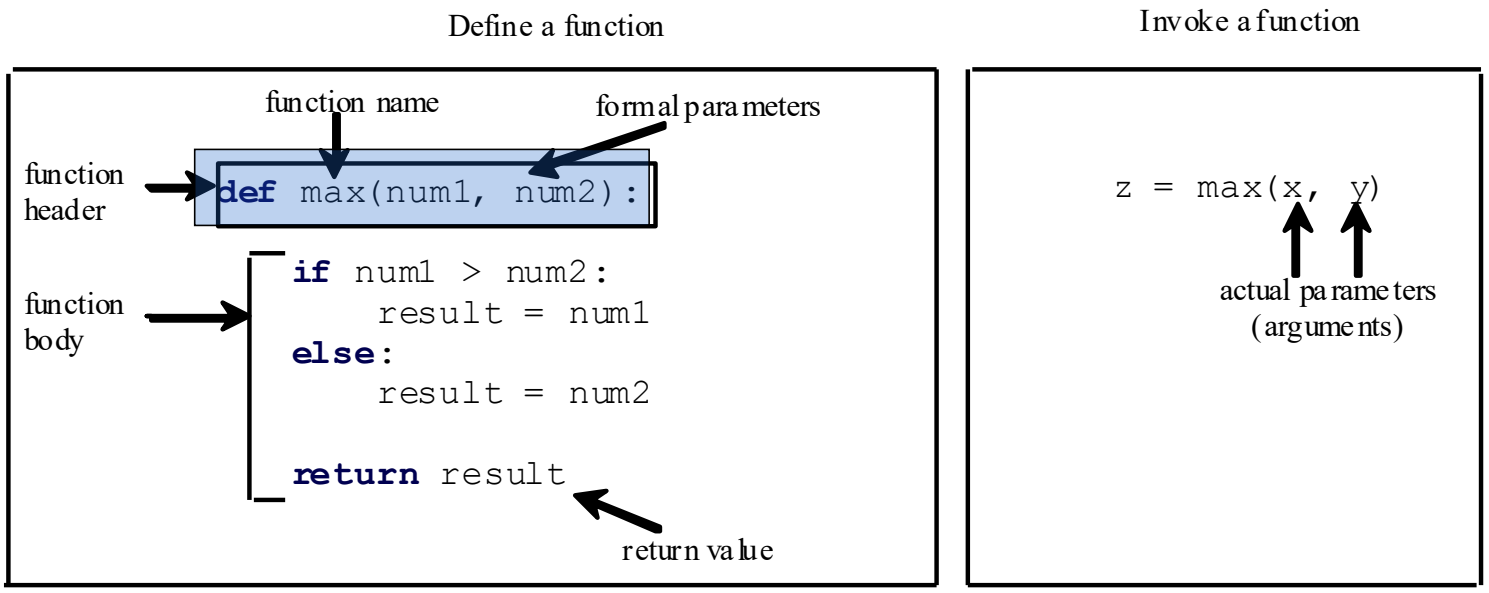
A function is a collection of statements that are grouped together to perform an operation.



UNIVERSITEIT GENT

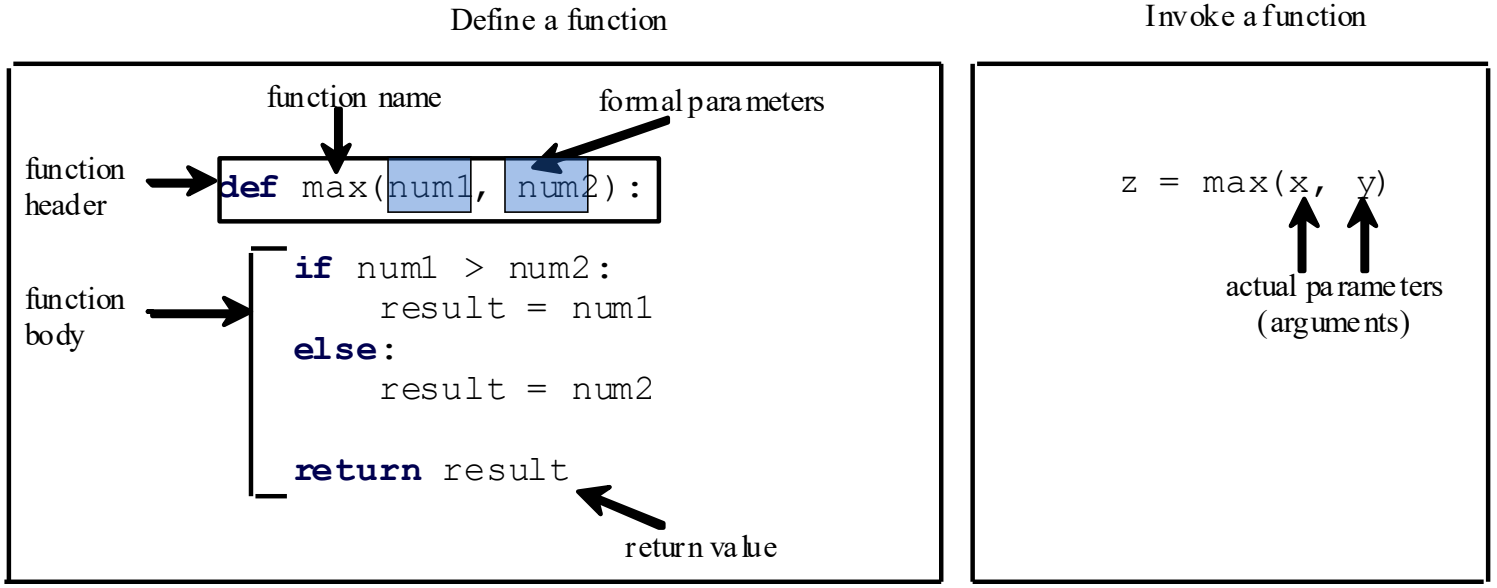
FUNCTION HEADER

A function contains a header and body. The header begins with the **def** keyword, followed by function's name and parameters, followed by a colon.



FORMAL PARAMETERS

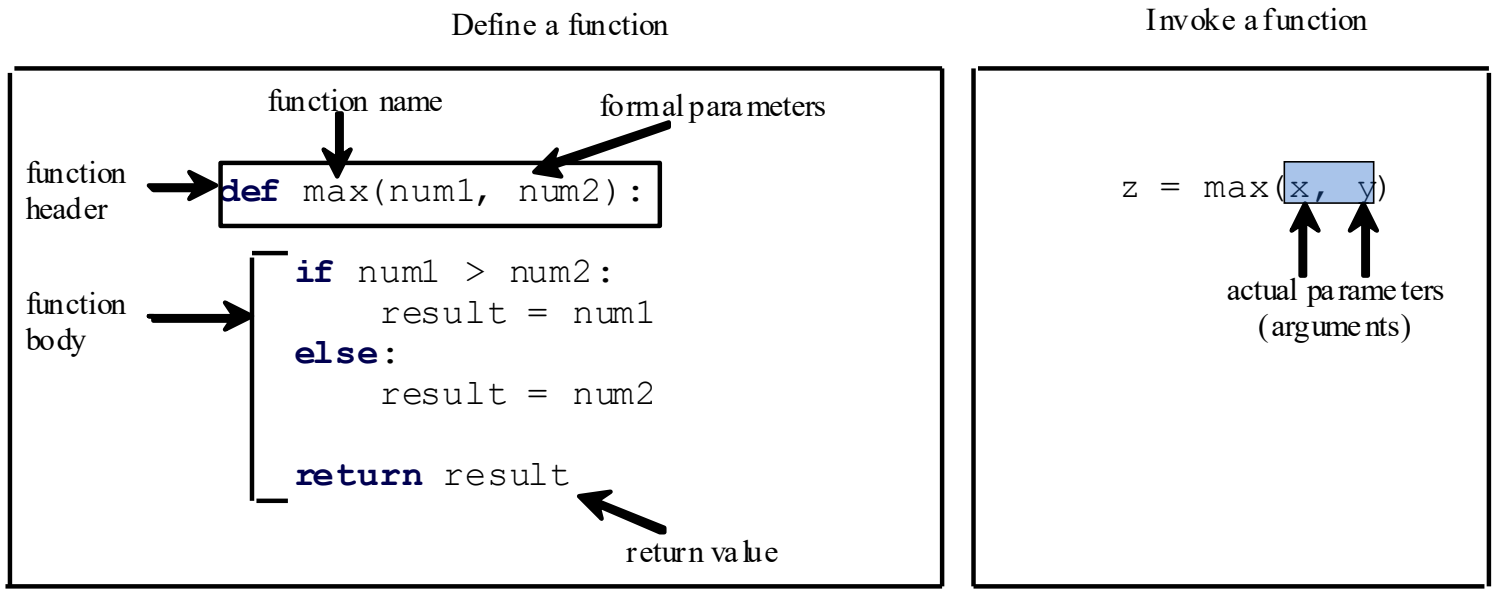
The variables defined in the function header are known as *formal parameters*.



UNIVERSITEIT GENT

ACTUAL PARAMETERS

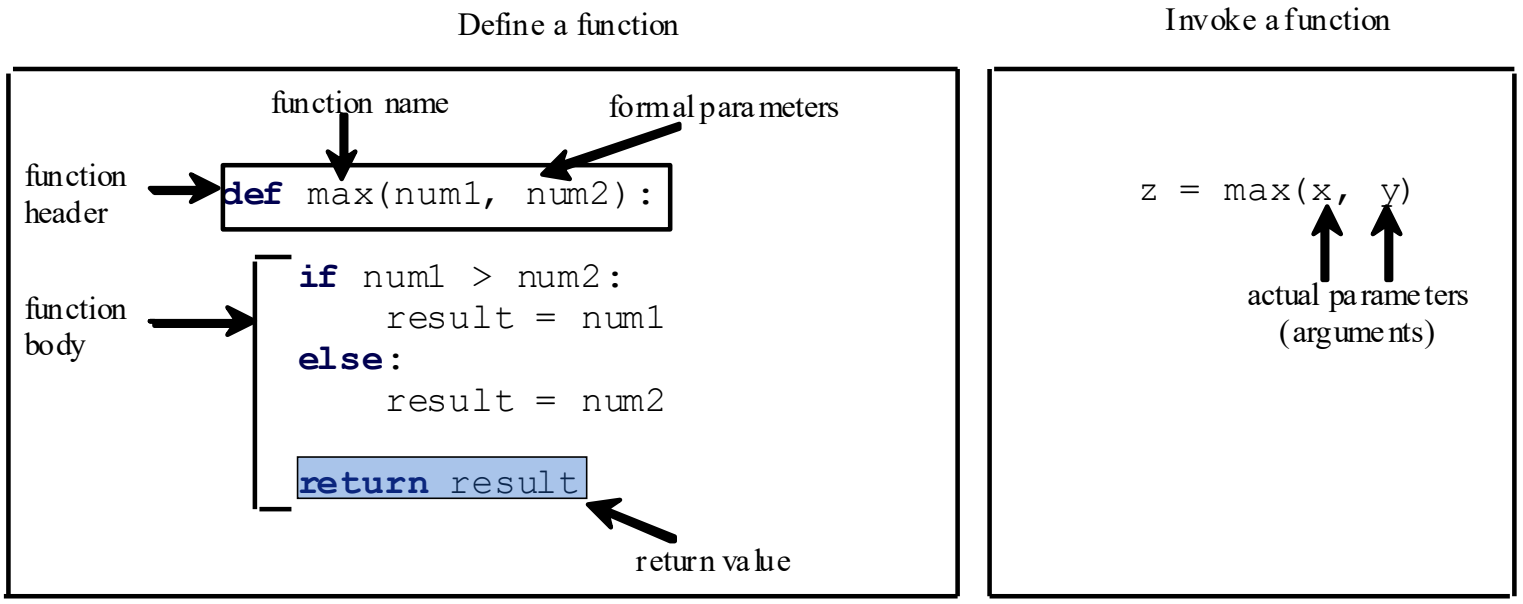
When a function is invoked, you pass a value to the parameter. This value is referred to as *actual parameter or argument*.



UNIVERSITEIT GENT

RETURN VALUE

A function may return a value using the return keyword.



UNIVERSITEIT GENT

CALLING FUNCTIONS

```
1  # Return the max between two numbers
2  def max(num1, num2):
3      if num1 > num2:
4          result = num1
5      else:
6          result = num2
7
8      return result # Return result
9
10 def main():
11     i = 5
12     j = 2
13     k = max(i, j) # Call the max function
14     print("The maximum between", i, "and", j, "is", k)
15
16 main() # Call the main function
```

Edit code

<< First | < Back | Step 1 of 14 | Forward > | Last >>

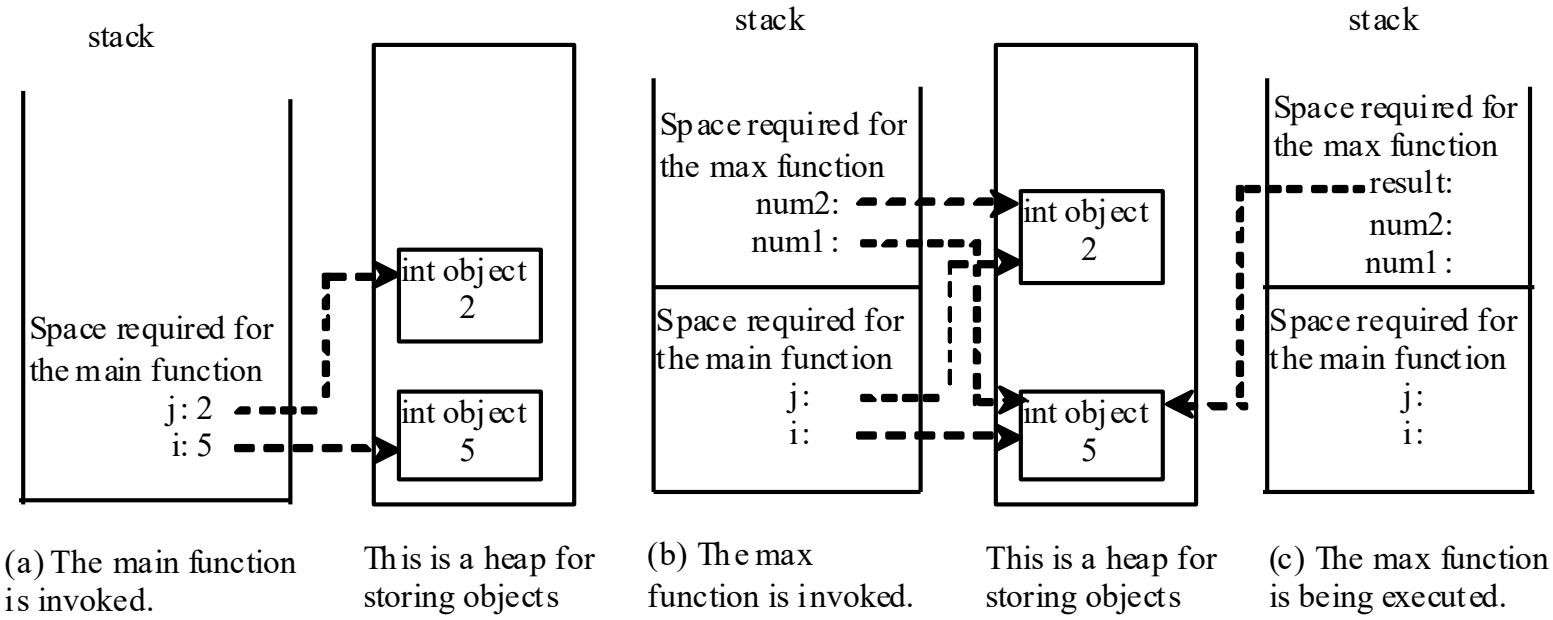
line that has just executed | next line to execute

Frames | Objects

Program output:

UNIVERSITEIT GENT

CALL STACKS



(a) The main function is invoked.

This is a heap for storing objects

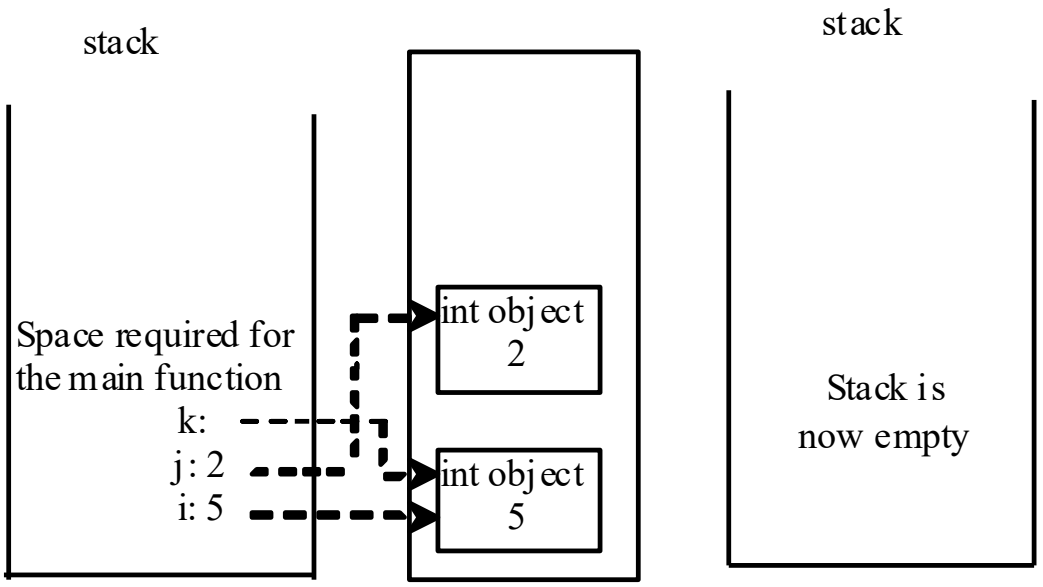
(b) The max function is invoked.

This is a heap for storing objects

(c) The max function is being executed.

UNIVERSITEIT GENT

CALL STACKS



(d) The max function is finished and the return value is sent to k.

This is a heap for storing objects

(e) The main function is finished.

UNIVERSITEIT GENT

FUNCTIONS WITH/WITHOUT RETURN VALUES

- This type of function does not return a value. The function performs some actions.

```
def getGrade(score):
    if score >= 90.0:
        return 'A'
    elif score >= 80.0:
        return 'B'
    elif score >= 70.0:
        return 'C'
    elif score >= 60.0:
        return 'D'
    else:
        return 'F'
```

```
def printGrade(score):
    if score >= 90.0:
        print('A')
    elif score >= 80.0:
        print('B')
    elif score >= 70.0:
        print('C')
    elif score >= 60.0:
        print('D')
    else:
        print('F')
```

THE NONE VALUE

A function that does not return a value is known as a *void* function in other programming languages such as Python, C++, and C#. In Python, such function returns a special value None

```
def sum(number1, number2):
    total = number1 + number2
print(sum(1, 2))
```

UNIVERSITEIT GENT

PASSING ARGUMENTS BY POSITIONS

Suppose you invoke the function using

nPrintln("Welcome to Python", 5)

What is the output?

Suppose you invoke the function using

nPrintln("Computer Science", 15)

What is the output?

What is wrong

nPrintln(4, "Computer Science")

```
def nPrintln(message, n):
    for i in range(0, n):
        print(message)
```

KEYWORD ARGUMENTS

What is wrong

nPrintln(4, “Computer Science”)

Is this OK?

nPrintln(n = 4, message = “Computer Science”)

```
def nPrintln(message, n):
    for i in range(0, n):
        print(message)
```

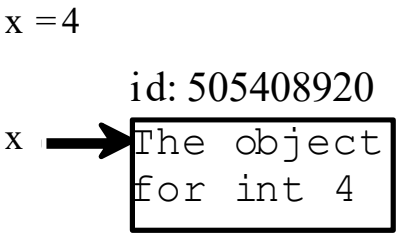
UNIVERSITEIT GENT

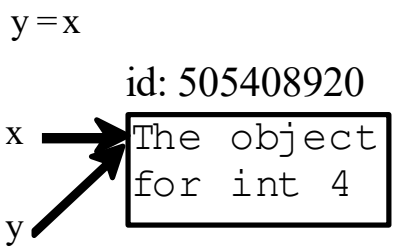
PASS BY VALUE

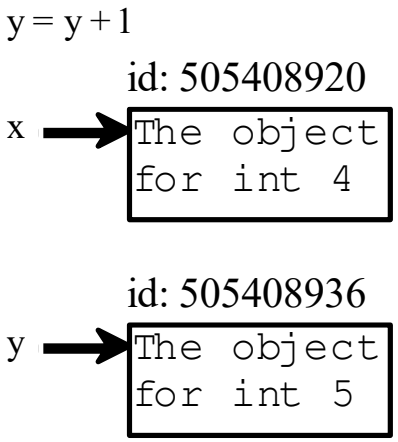
- In Python, all data are objects. A variable for an object is actually a reference to the object. When you invoke a function with a parameter, the reference value of the argument is passed to the parameter. This is referred to as *pass-by-value*. For simplicity, we say that the value of an argument is passed to a parameter when invoking a function. Precisely, the value is actually a reference value to the object.
- If the argument is a number or a string, the argument is not affected, regardless of the changes made to the parameter inside the function.

Increment

PASS BY VALUE







MODULARIZING CODE

Functions can be used to reduce redundant coding and enable code reuse. Functions can also be used to modularize code and improve the quality of the program.

GCDFunction

TestGCDFunction

PrimeNumberFunction

SCOPE OF VARIABLES

- Scope: the part of the program where the variable can be referenced.
- A variable created inside a function is referred to as a *local variable*. Local variables can only be accessed inside a function. The scope of a local variable starts from its creation and continues to the end of the function that contains the variable.
- In Python, you can also use *global variables*. They are created outside all functions and are accessible to all functions in their scope.

EXAMPLE 1

```
globalVar = 1
def f1():
    localVar = 2
    print(globalVar)
    print(localVar)
f1()
print(globalVar)
print(localVar)  # Out of scope. This gives an error
```

UNIVERSITEIT
GENT

EXAMPLE 2

```
x = 1
def f1():
    x = 2
    print(x) # Displays 2
f1()
print(x) # Displays 1
```

UNIVERSITEIT
GENT

EXAMPLE 3

```
x = eval(input("Enter a number: "))
if x > 0:
    y = 4
print(y) # This gives an error if y is not created
```

EXAMPLE 4

```
sum = 0
for i in range(5):
    sum += i
print(i)
```

EXAMPLE 5

```
x = 1
def increase():
    global x
    x = x + 1
    print(x) # Displays 2
increase()
print(x) # Displays 2
```

UNIVERSITEIT
GENT

DEFAULT ARGUMENTS

Python allows you to define functions with default argument values. The default values are passed to the parameters when a function is invoked without the arguments.

DefaultArgumentDemo

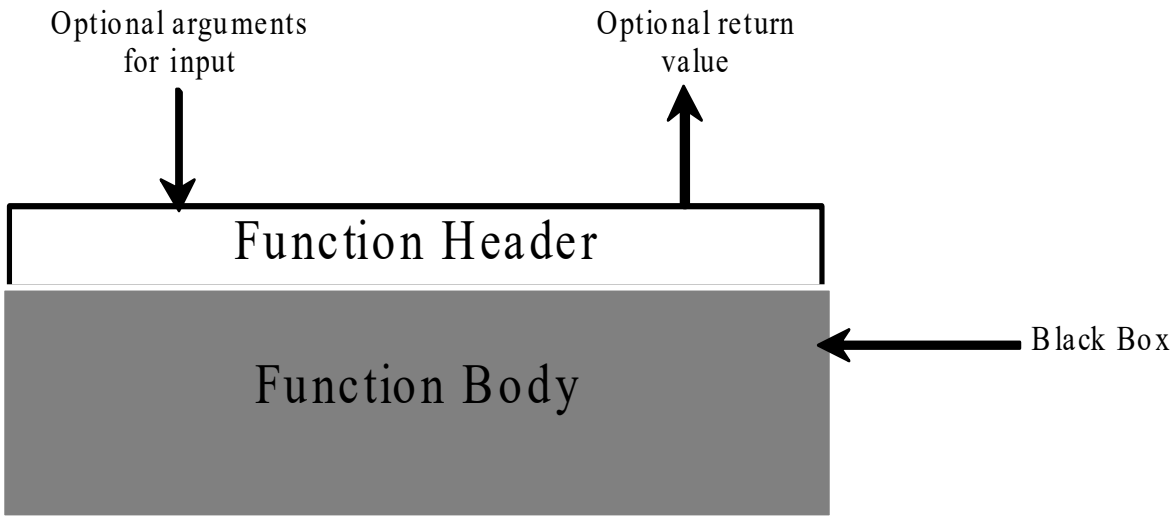
RETURNING MULTIPLE VALUES

Python allows a function to return multiple values. Listing 6.9 defines a function that takes two numbers and returns them in non-descending order.

MultipleReturnValueDemo

FUNCTION ABSTRACTION

- You can think of the function body as a black box that contains the detailed implementation for the function.



UNIVERSITEIT GENT

BENEFITS OF FUNCTIONS

- Write a function once and reuse it anywhere.
- Information hiding. Hide the implementation from the user.
- Reduce complexity.

STEPWISE REFINEMENT

The concept of function abstraction can be applied to the process of developing programs. When writing a large program, you can use the “divide and conquer” strategy, also known as stepwise refinement, to decompose it into subproblems. The subproblems can be further decomposed into smaller, more manageable problems.

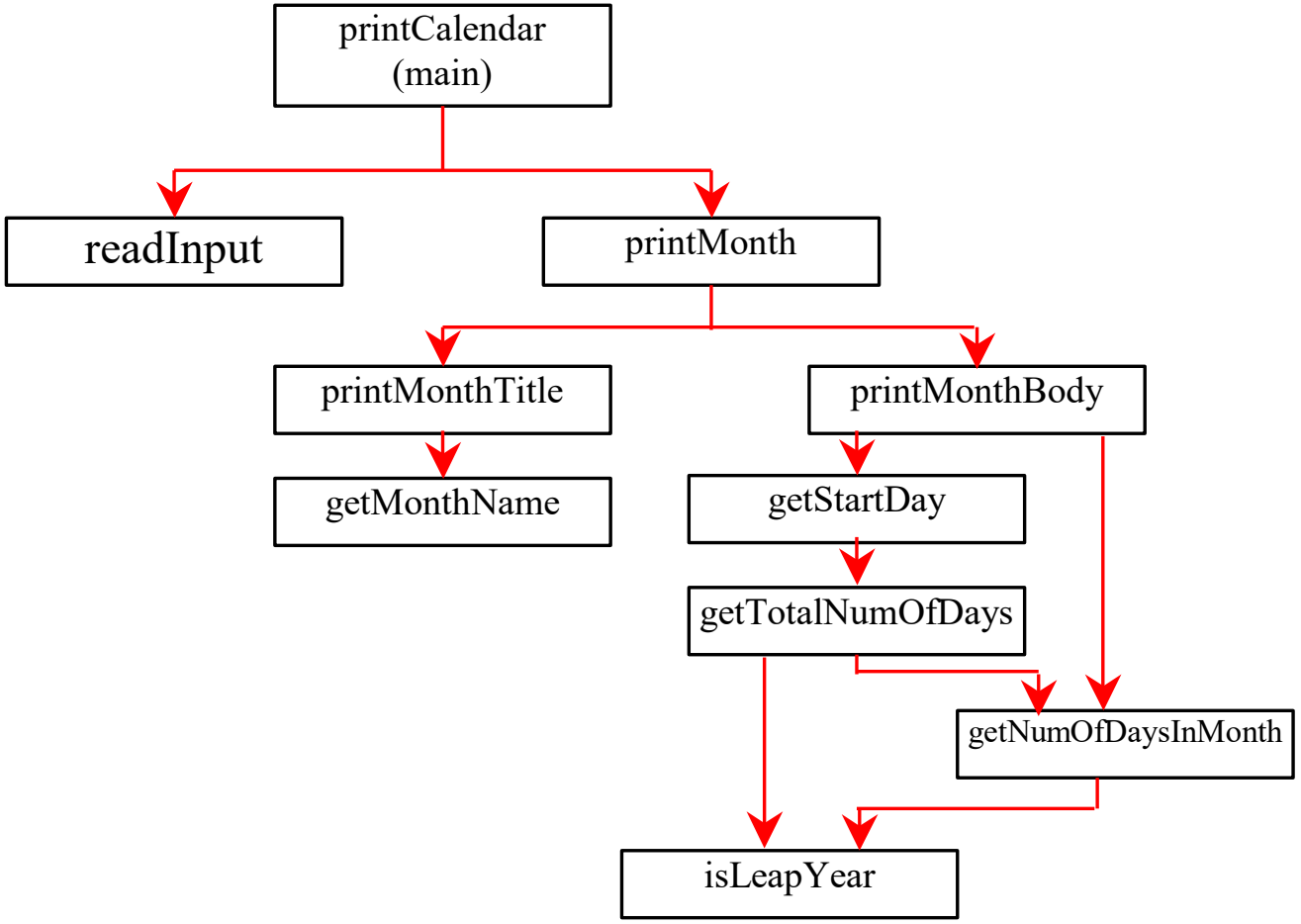
PRINTCALENDER CASE STUDY

Let us use the PrintCalendar example to demonstrate the stepwise refinement approach.

```
Enter full year (e.g., 2001): 2022
Enter month as number between 1 and 12: 6
          June   2022
-----------------------------
 Sun Mon Tue Wed Thu Fri Sat
                   1   2   3   4
   5   6   7   8   9  10  11
  12  13  14  15  16  17  18
  19  20  21  22  23  24  25
  26  27  28  29  30
```

UNIVERSITEIT GENT

DESIGN DIAGRAM



UNIVERSITEIT
GENT

IMPLEMENTATION: TOP-DOWN

Top-down approach is to implement one function in the structure chart at a time from the top to the bottom. Stubs can be used for the functions waiting to be implemented. A stub is a simple but incomplete version of a function. The use of stubs enables you to test invoking the function from a caller. Implement the main function first and then use a stub for the printMonth function. For example, let printMonth display the year and the month in the stub. Thus, your program may begin like this:

A Skeleton for PrintCalendar

IMPLEMENTATION: BOTTOM-UP

Bottom-up approach is to implement one function in the structure chart at a time from the bottom to the top. For each function implemented, write a test program to test it. Both top-down and bottom-up functions are fine. Both approaches implement the functions incrementally and help to isolate programming errors and makes debugging easy. Sometimes, they can be used together.

CHAPTER 7 LISTS

UNIVERSITEIT
GENT

OPENING PROBLEM

Read one hundred numbers, compute their average, and find out how many numbers are above the average.

OBJECTIVES

- To describe why lists are useful in programming (§7.1).
- To create lists (§7.2.1).
- To invoke list's append, insert, extend, remove, pop, index, count, sort, reverse methods (§7.2.2).
- To use the len, min/max, sum, and random.shuffle functions for a list (§7.2.3).
- To access list elements using indexed variables (§7.2.4).
- To obtain a sublist using the slicing operator [start:end] (§7.2.5).
- To use +, *, and in/not in operators on lists (§7.2.6).
- To traverse elements in a list using a for-each loop (§7.2.7).
- To create lists using list comprehension (§7.2.8).
- To compare lists using comparison operators (§7.2.9).
- To split a string to a list using the str's split method (§7.2.10).
- To use lists in the application development (§§7.3–7.5).
- To copy contents from one list to another (§7.6).
- To develop and invoke functions with list arguments and return value (§7.7–7.9).
- To search elements using the linear (§7.7.1) ~~or binary (§7.7.2) search algorithm~~.
- ~~To sort a list using the selection sort (§7.11.1)~~
- ~~To sort a list using the insertion sort (§7.11.2).~~
- ~~To develop the bouncing ball animation using a list (§7.12).~~

UNIVERSITEIT
GENT

CREATING LISTS

- Creating list using the list class

```
list1 = list() # Create an empty list
list2 = list([2, 3, 4]) # Create a list with elements 2, 3, 4
list3 = list(["red", "green", "blue"]) # Create a list with strings
list4 = list(range(3, 6)) # Create a list with elements 3, 4, 5
list5 = list("abcd") # Create a list with characters a, b, c
```

- For convenience, you may create a list using the following syntax:

```
list1 = [] # Same as list()
list2 = [2, 3, 4] # Same as list([2, 3, 4])
list3 = ["red", "green"] # Same as list(["red", "green"])
```

LIST METHODS

list	
append(x: object): None	Add an item x to the end of the list.
insert(index: int, x: object): None	Insert an item x at a given index. Note that the first element in the list has index 0.
remove(x: object): None	Remove the first occurrence of the item x from the list.
index(x: object): int	Return the index of the item x in the list.
count(x: object): int	Return the number of times item x appears in the list.
sort(): None	Sort the items in the list.
reverse(): None	Reverse the items in the list.
extend(l: list): None	Append all the items in L to the list.
pop([i]): object	Remove the item at the given position and return it. The square bracket denotes that parameter is optional. If no index is specified, list.pop() removes and returns the last item in the list.

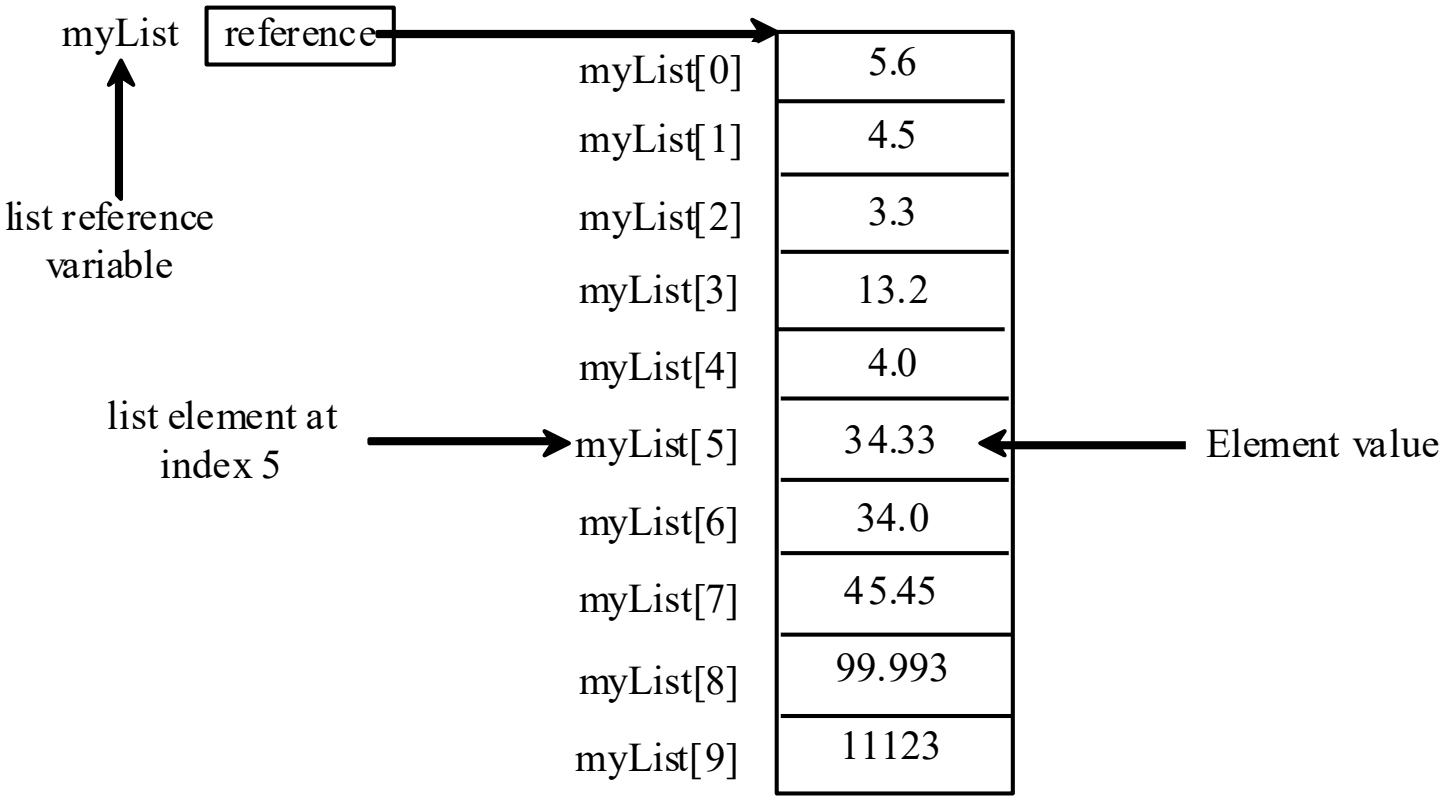
UNIVERSITEIT GENT

FUNCTIONS FOR LISTS

```
>>> list1 = [2, 3, 4, 1, 32]
>>> len(list1)
5
>>> max(list1)
32
>>> min(list1)
1
>>> sum(list1)
42
>>> import random
>>> random.shuffle(list1) # Shuffle the items in the list
>>> list1
[4, 1, 2, 32, 3]
```

INDEXER OPERATOR []

myList = [5.6, 4.5, 3.3, 13.2, 4.0, 34.33, 34.0, 45.45, 99.993, 11123]



UNIVERSITEIT GENT

THE +, *, [:], AND IN OPERATORS

```
>>> list1 = [2, 3]
>>> list2 = [1, 9]
>>> list3 = list1 + list2
>>> list3
[2, 3, 1, 9]
>>> list3 = 2 * list1
>>> list3
[2, 3, 2, 3, 2, 3]
>>> list4 = list3[2 : 4]
>>> list4
[2, 3]
```

```
>>> list1 = [2, 3, 5, 2, 33, 21]
>>> list1[-1]
21
>>> list1[-3]
2
>>> list1 = [2, 3, 5, 2, 33, 21]
>>> 2 in list1
True
>>> list1 = [2, 3, 5, 2, 33, 21]
>>> 2.5 in list1
False
```

OFF-BY-ONE ERROR

```
i = 0
while i <= len(lst):
    print(lst[i])
    i += 1
```

UNIVERSITEIT
GENT

LIST COMPREHENSION

List comprehensions provide a concise way to create items from sequence. A list comprehension consists of brackets containing an expression followed by a for clause, then zero or more for or if clauses. The result will be a list resulting from evaluating the expression. Here are some examples:

```
>>> list1 = [x for x range(0, 5)] # Returns a list of 0, 1, 2, 4
>>> list1
[0, 1, 2, 3, 4]
>>> list2 = [0.5 * x for x in list1]
>>> list2
[0.0, 0.5, 1.0, 1.5, 2.0]
>>> list3 = [x for x in list2 if x < 1.5]
>>> list3
[0.0, 0.5, 1.0]
```

UNIVERSITEIT GENT

COMPARING LISTS

```
>>>list1 = ["green", "red", "blue"]
>>>list2 = ["red", "blue", "green"]
>>>list2 == list1
False
>>>list2 != list1
True
>>>list2 >= list1
False
>>>list2 > list1
False
>>>list2 < list1
True
>>>list2 <= list1
True
```

UNIVERSITEIT GENT

SPLITTING A STRING TO A LIST

```
items = "Welcome to the US".split()
print(items)
['Welcome', 'to', 'the', 'US']
```

- items = "34#13#78#45".split("#")
- print(items)
- ['34', '13', '78', '45']

UNIVERSITEIT GENT

ANALYZE NUMBERS

Read one hundred numbers, compute their average, and find out how many numbers are above the average.

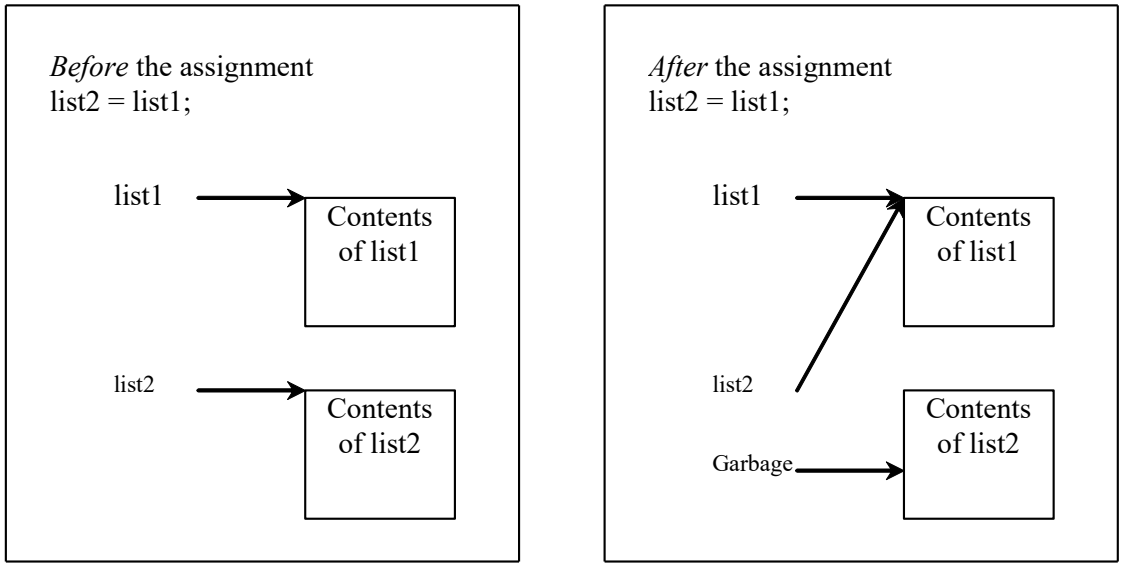
AnalyzeNumbers

UNIVERSITEIT
GENT

COPYING LISTS

Often, in a program, you need to duplicate a list or a part of a list. In such cases you could attempt to use the assignment statement (=), as follows:

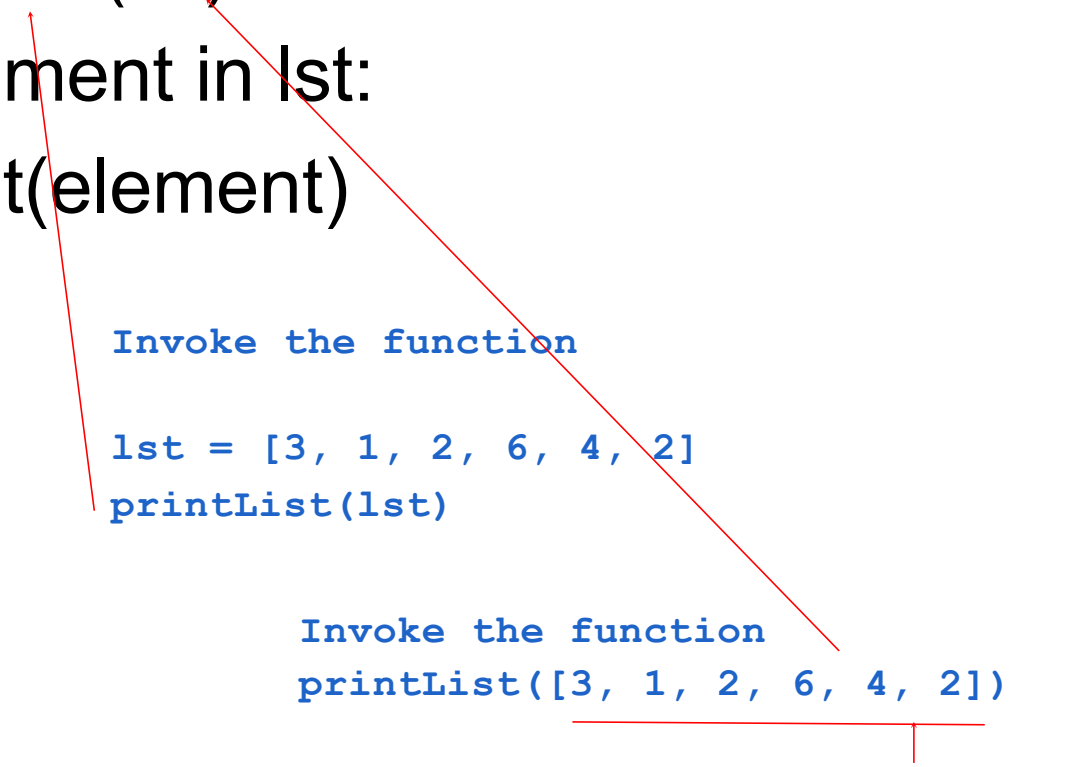
list2 = list1;



UNIVERSITEIT GENT

PASSING LISTS TO FUNCTIOS

```
def printList(lst):
    for element in lst:
        print(element)
```



PASS BY VALUE

Python uses pass-by-value to pass arguments to a function. There are important differences between passing the values of variables of numbers and strings and passing lists.

- Immutable objects
- Changeable objects

PASS BY VALUE (IMMUTABLE OBJECTS)

For an argument of a number or a string, the original value of the number and string outside the function is not changed, because numbers and strings are immutable in Python.

PASS BY VALUE (CHANGEABLE OBJECTS)

For an argument of a list, the value of the argument is a reference to a list; this reference value is passed to the function. Semantically, it can be best described as pass-by-sharing, i.e., the list in the function is the same as the list being passed. So if you change the list in the function, you will see the change outside the function.

SIMPLE EXAMPLE

```
def main():
    x = 1 # x represents an int value
    y = [1, 2, 3] # y represents a list
    m(x, y) # Invoke f with arguments x and y
    print("x is " + str(x))
    print("y[0] is " + str(y[0]))

def m(number, numbers):
    number = 1001 # Assign a new value to number
    numbers[0] = 5555 # Assign a new value to numbers[0]

main()
```

SUBTLE ISSUES REGARDING DEFAULT ARGUMENTS

```
def add(x, lst = []):
    if not(x in lst):
        lst.append(x)
    return lst
list1 = add(1)
print(list1)
list2 = add(2)
print(list2)
list3 = add(3, [11, 12, 13, 14])
print(list3)
list4 = add(4)
print(list4)
```

default value is created only once.

Output

```
[1]
[1, 2]
[11, 12, 13, 14]
[1, 2, 4]
```

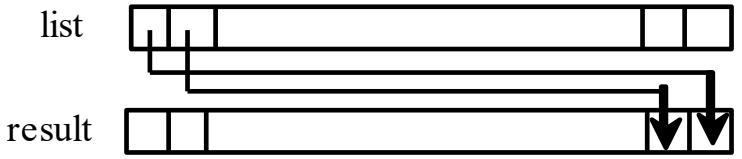
UNIVERSITEIT GENT

RETURNING A LIST FROM A FUNCTION

list1 = [1, 2, 3, 4, 5, 6]

list2 = reverse(list1)

```
def reverse(list):
    result = []

    for element in list:
        result.insert(0, element)

    return result
```



Note that list already has the reverse method list.reverse()

ISSUES: LIST AS A DEFAULT ARGUMENT

```
def add(x, lst = []):
    if x not in lst:
        lst.append(x)

    return lst

def main():
```

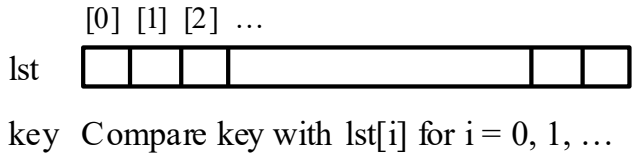
```
def add(x, lst = None):
    if lst == None:
        lst = []
    if not(x in lst):
        lst.append(x)

    return lst
```

UNIVERSITEIT
GENT

SEARCHING LISTS

Searching is the process of looking for a specific element in a list; for example, discovering whether a certain score is included in a list of scores. Searching is a common task in computer programming.

```
# The function for finding a key in the list
def linearSearch(lst, key):
    for i in range(0, len(lst)):
        if key == lst[i]:
            return i

    return -1
```



UNIVERSITEIT GENT

CHAPTER 8

LISTS FOR MULTI-DIMENSIONAL DATA

UNIVERSITEIT
GENT

MOTIVATIONS

Distance Table (in miles)

	Chicago	Boston	New York	Atlanta	Miami	Dallas	Houston
Chicago	0	983	787	714	1375	967	1087
Boston	983	0	214	1102	1763	1723	1842
New York	787	214	0	888	1549	1548	1627
Atlanta	714	1102	888	0	661	781	810
Miami	1375	1763	1549	661	0	1426	1187
Dallas	967	1723	1548	781	1426	0	239
Houston	1087	1842	1627	810	1187	239	0

MOTIVATIONS

```
distances = [
        [0, 983, 787, 714, 1375, 967, 1087],
        [983, 0, 214, 1102, 1763, 1723, 1842],
        [787, 214, 0, 888, 1549, 1548, 1627],
        [714, 1102, 888, 0, 661, 781, 810],
        [1375, 1763, 1549, 661, 0, 1426, 1187],
        [967, 1723, 1548, 781, 1426, 0, 239],
        [1087, 1842, 1627, 810, 1187, 239, 0]
]
```

UNIVERSITEIT
GENT

OBJECTIVES

- To give examples of representing data using two-dimensional lists (§8.1).
- To access elements in a two-dimensional list using row and column indexes (§8.2).
- To program common operations for two-dimensional lists (displaying lists, summing all elements, finding min and max elements, and random shuffling) (§8.2).
- To pass two-dimensional lists to functions (§8.3).
- To write a program for grading multiple-choice questions using two-dimensional lists (§8.4).
- To solve the closest-pair problem using two-dimensional lists (§§8.5-8.6).
- To check a Sudoku solution using two-dimensional lists (§§8.7-8.8).
- To use multidimensional lists (§8.9).

PROCESSING TWO-DIMENSIONAL LISTS

You can view a two-dimensional list as a list that consists of rows. Each row is a list that contains the values. The rows can be accessed using the index, conveniently called a row index. The values in each row can be accessed through another index, conveniently called a column index.

```
matrix = [
    [1, 2, 3, 4, 5],
    [6, 7, 0, 0, 0],
    [0, 1, 0, 0, 0],
    [1, 0, 0, 0, 8],
    [0, 0, 9, 0, 3],
]
```

	[0]	[1]	[2]	[3]	[4]
[0]	1	2	3	4	5
[1]	6	7	0	0	0
[2]	0	1	0	0	0
[3]	1	0	0	0	8
[4]	0	0	9	0	3

```
matrix[0] is [1, 2, 3, 4, 5]
matrix[1] is [6, 7, 0, 0, 0]
matrix[2] is [0, 1, 0, 0, 0]
matrix[3] is [1, 0, 0, 0, 8]
matrix[4] is [0, 0, 9, 0, 3]

matrix[0][0] is 1
matrix[4][4] is 3
```

UNIVERSITEIT GENT

PROCESSING TWO-DIMENSIONAL LISTS

See the examples in code.

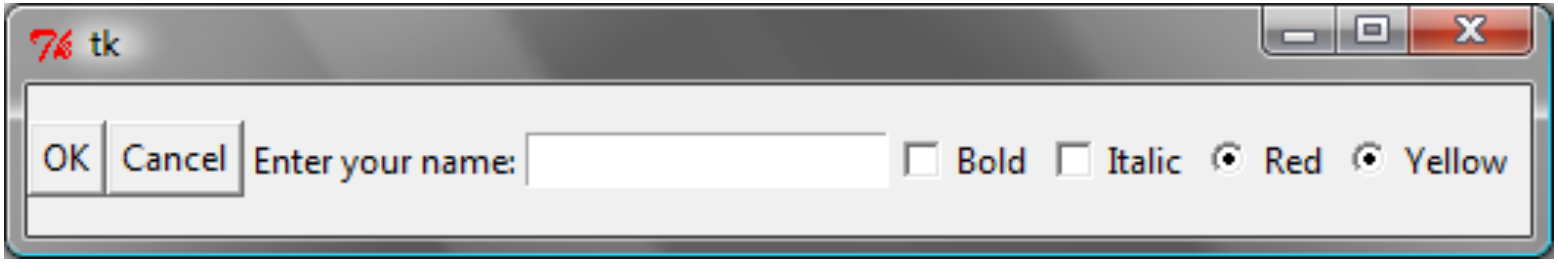
1. (Initializing lists with input values)
2. (Initializing lists with random values)
3. (Printing lists)
4. (Summing all elements)
5. (Summing all elements by column)
6. (Which row has the largest sum)
7. (Random shuffling)

CHAPTER 9 OBJECT-ORIENTED PROGRAMMING

UNIVERSITEIT GENT

MOTIVATIONS

After learning the preceding chapters, you are capable of solving many programming problems using selections, loops, and functions. However, these Python features are not sufficient for developing graphical user interfaces and large scale software systems. Suppose you want to develop a graphical user interface as shown below. How do you program it?



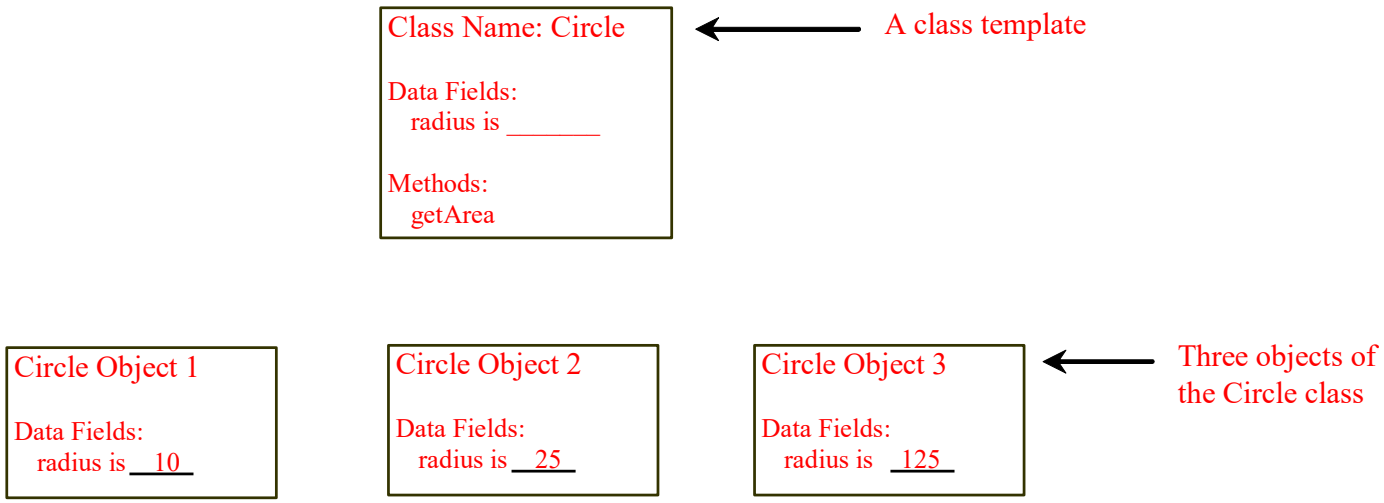
OBJECTIVES

- To describe objects and classes, and use classes to model objects (§9.2).
- To define classes with data fields and methods (§9.2.1).
- To construct an object using a constructor that invokes the initializer to create and initialize data fields (§9.2.2).
- To access the members of objects using the dot operator (.) (§9.2.3).
- To reference an object itself with the self parameter (§9.2.4).
- To use UML graphical notation to describe classes and objects (§9.3).
- To use the datetime class from the Python library (§9.4).
- To distinguish between immutable and mutable objects (§9.5).
- To hide data fields to prevent data corruption and make classes easy to maintain (§9.6).
- To apply class abstraction and encapsulation to software development (§9.7).
- To explore the differences between the procedural paradigm and the object-oriented paradigm (§9.8).
- ~~To define special methods for operators (§9.9).~~
- ~~To design the Rational class for representing rational numbers (§9.10).~~

OO PROGRAMMING CONCEPTS

Object-oriented programming (OOP) involves programming using objects. An *object* represents an entity in the real world that can be distinctly identified. For example, a student, a desk, a circle, a button, and even a loan can all be viewed as objects. An object has a unique identity, state, and behaviors. The *state* of an object consists of a set of *data fields* (also known as *properties*) with their current values. The *behavior* of an object is defined by a set of methods.

OBJECTS



An object has both a state and behavior. The state defines the object, and the behavior defines what the object does.

CLASSES

A Python class uses variables to store data fields and defines methods to perform actions. Additionally, a class provides a special type method, known as *initializer*, which is invoked to create a new object. An initializer can perform any action, but initializer is designed to perform initializing actions, such as creating the data fields of objects.

```
class ClassName:
    initializer
    methods
```

Circle

TestCircle

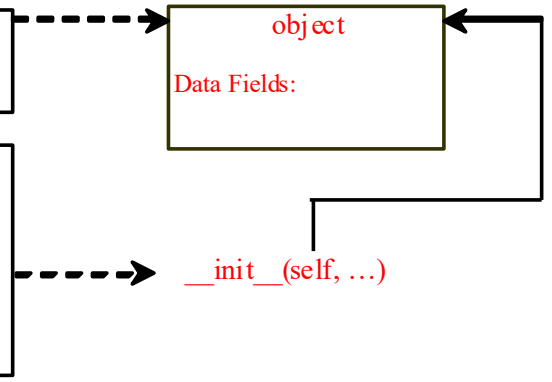
UNIVERSITEIT GENT

CONSTRUCTING OBJECTS

Once a class is defined, you can create objects from the class by using the following syntax, called a *constructor*:

```
ClassName(arguments)
```

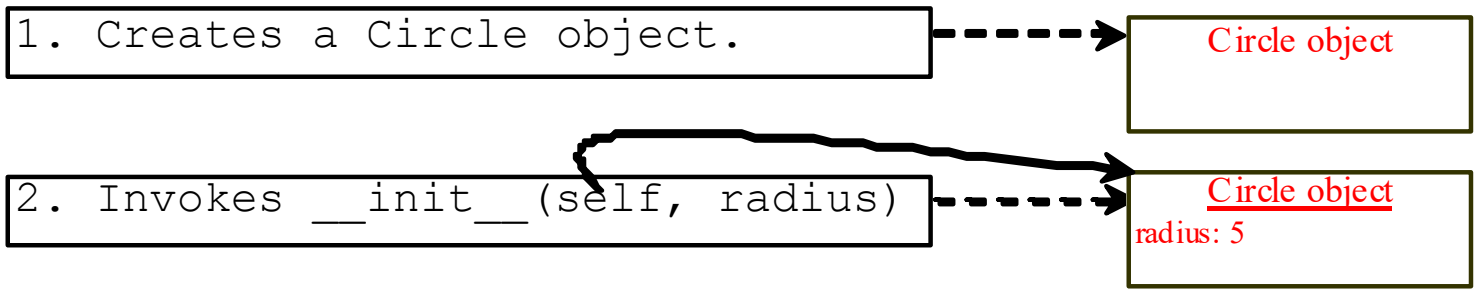
1. It creates an object in the memory for the class.
2. It invokes the class's `__init__` method to initialize the object. The self parameter in the `__init__` method is automatically set to reference the object that was just created.



UNIVERSITEIT GENT

CONSTRUCTING OBJECTS

The effect of constructing a Circle object using Circle(5) is shown below:



UNIVERSITEIT GENT

INSTANCE METHODS

Methods are functions defined inside a class. They are invoked by objects to perform actions on the objects. For this reason, the methods are also called *instance methods* in Python. You probably noticed that all the methods including the __init__ have the first parameter **self**, which refers to the object that invokes the method. You can use any name for this parameter. But by convention, **self** is used.

ACCESSING OBJECTS

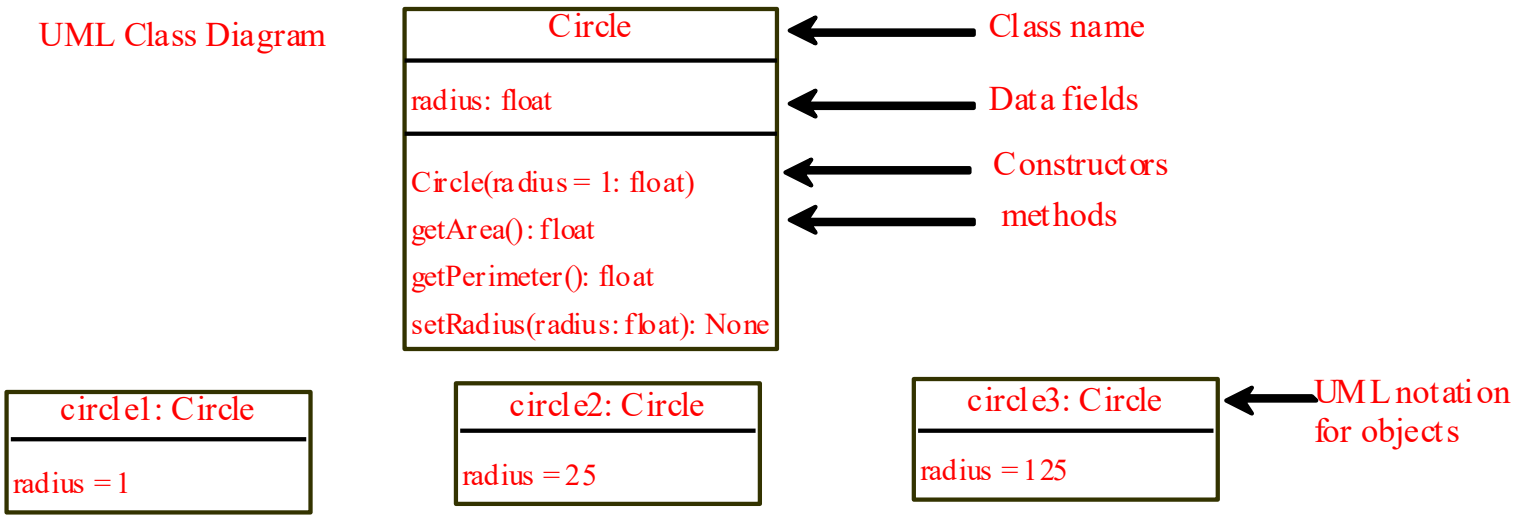
After an object is created, you can access its data fields and invoke its methods using the dot operator (.), also known as the *object member access operator*. For example, the following code accesses the radius data field and invokes the getPerimeter and getArea methods.

```
>>> from Circle import Circle
>>> c = Circle(5)
>>> c.getPerimeter()
31.41592653589793
>>> c.radius = 10
>>> c.getArea()
314.1592653589793
```

WHY SELF?

- Note that the first parameter is special. It is used in the implementation of the method, but not used when the method is called. So, what is this parameter self for? Why does Python need it?
- self is a parameter that represents an object. Using self, you can access instance variables in an object. Instance variables are for storing data fields. Each object is an instance of a class. Instance variables are tied to specific objects. Each object has its own instance variables. You can use the syntax self.x to access the instance variable x for the object self in a method.

UML CLASS DIAGRAM



UNIVERSITEIT
GENT

EXAMPLE: DEFINING CLASSES AND CREATING OBJECTS

TV	
channel: int	The current channel (1 to 120) of this TV.
volumeLevel: int	The current volume level (1 to 7) of this TV.
on: bool	Indicates whether this TV is on/off.
TV()	Constructs a default TV object.
turnOn(): None	Turns on this TV.
turnOff(): None	Turns off this TV.
getChannel(): int	Returns the channel for this TV.
setChannel(channel: int): None	Sets a new channel for this TV.
getVolume(): int	Gets the volume level for this TV.
setVolume(volumeLevel: int): None	Sets a new volume level for this TV.
channelUp(): None	Increases the channel number by 1.
channelDown(): None	Decreases the channel number by 1.
volumeUp(): None	Increases the volume level by 1.
volumeDown(): None	Decreases the volume level by 1.

TV

TestTV

UNIVERSITEIT GENT

IMMUTABLE OBJECTS VS. MUTABLE OBJECTS

```
def printAreas(c, times):

    print("Radius \t\tArea")
    while times >= 1:
        print(c.radius, "\t\t", c.getArea())
        c.radius = c.radius + 1 # increase radius
        times -= 1
```

```
def main():
    # Create a Circle object with radius 1
    myCircle = Circle()

    # Print areas for radius 1, 2, 3, 4, and 5.
    n = 5
    printAreas(myCircle, n)

    # Display myCircle.radius and times
    print("\nRadius is", myCircle.radius)
    print("n is", n)
```

UNIVERSITEIT GENT

THE DATETIME CLASS

```
from datetime import datetime
d = datetime.now()
print("Current year is " + str(d.year))
print("Current month is " + str(d.month))
print("Current day of month is " + str(d.day))
print("Current hour is " + str(d.hour))
print("Current minute is " + str(d.minute))
print("Current second is " + str(d.second))
```

DATA FIELD ENCAPSULATION

- To protect data.
- To make class easy to maintain.
- To prevent direct modifications of data fields, don't let the client directly access data fields. This is known as *data field encapsulation*. This can be done by defining private data fields. In Python, the private data fields are defined with two leading underscores. You can also define a private method named with two leading underscores.

UNIVERSITEIT GENT

CircleWithPrivateRadius

DATA FIELD ENCAPSULATION

```
>>> from CircleWithPrivateRadius import Circle
>>> c = Circle(5)
>>> c.__radius
AttributeError: 'Circle' object has no attribute
'__radius'
>>> c.getRadius()
 5
```

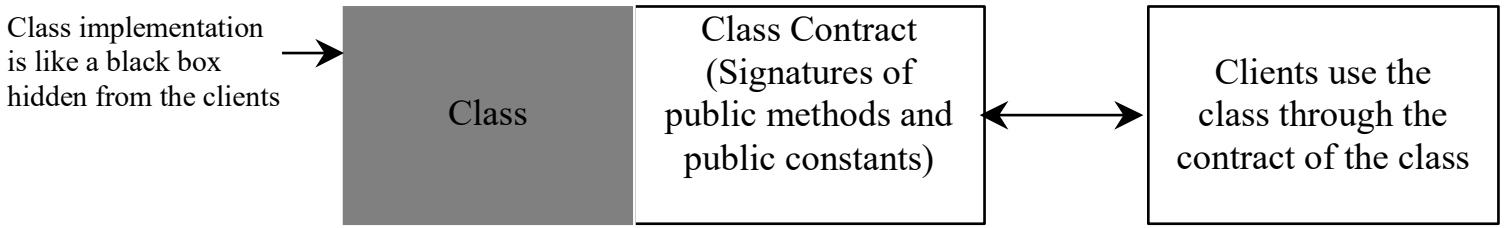
UNIVERSITEIT
GENT

DESIGN GUIDE

If a class is designed for other programs to use, to prevent data from being tampered with and to make the class easy to maintain, define data fields private. If a class is only used internally by your own program, there is no need to encapsulate the data fields.

CLASS ABSTRACTION AND ENCAPSULATION

Class abstraction means to separate class implementation from the use of the class. The creator of the class provides a description of the class and let the user know how the class can be used. The user of the class does not need to know how the class is implemented. The detail of implementation is encapsulated and hidden from the user.

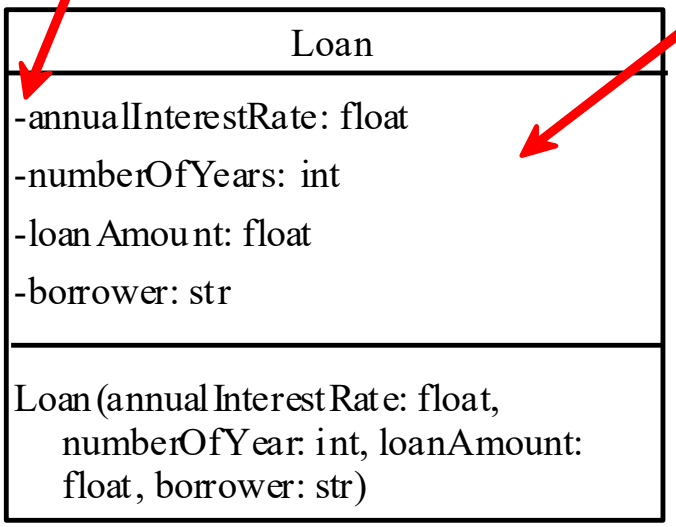


UNIVERSITEIT GENT

DESIGNING THE LOAN CLASS

The – sign denotes a private data field.

The get methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.



The annual interest rate of the loan (default: 2.5).

The number of years for the loan (default: 1)

The loan amount (default: 1000).

The borrower of this loan.

Constructs a Loan object with the specified annual interest rate, number of years, loan amount, and borrower.

Loan

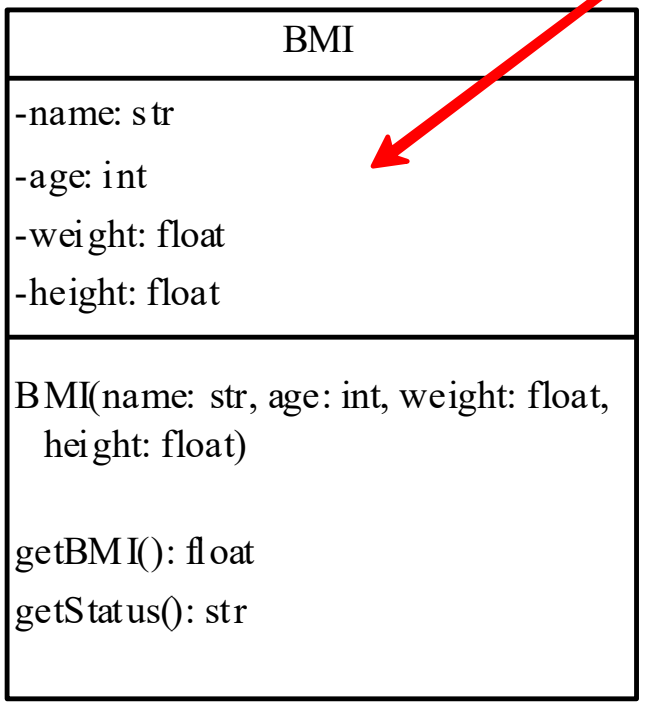
TestLoanClass

UNIVERSITEIT GENT

OBJECT-ORIENTED THINKING

This book's approach is to teach problem solving and fundamental programming techniques before object-oriented programming. This section will show how procedural and object-oriented programming differ. You will see the benefits of object-oriented programming and learn to use it effectively. We will use several examples in the rest of the chapter to illustrate the advantages of the object-oriented approach. The examples involve designing new classes and using them in applications.

THE BMI CLASS



The get methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

The name of the person.

The age of the person.

The weight of the person in pounds.

The height of the person in inches.

Creates a BMI object with the specified name, weight, height, and a default age 20.

Returns the BMI

Returns the BMI status (e.g., normal, overweight, etc.)

UNIVERSITEIT GENT

BMI

UseBMIClass

THE DATETIME CLASS

datetime	
year: int month: int day: int hour: int minute: int second: int microsecond: int	The year, month, day, hour, minute, second, and microsecond in this datatime object.
datetime(year, month, day, hour = 0, minute = 0, second = 0, microsecond = 0)	Creates a datetime object using the specified year, month, day, hour, minute, second, and microsecond.
now(): datetime	Returns a datetime object for the current time.
fromtimestamp(timestamp): datetime	Returns a datetime object from the specified timestamp in seconds.
timestamp(): int	Returns the timestamp in seconds in this datetime object.

UNIVERSITEIT GENT

DatetimeDemo

PROCEDURAL VS. OBJECT-ORIENTED

In procedural programming, data and operations on the data are separate, and this methodology requires sending data to methods. Object-oriented programming places data and the operations that pertain to them in an object. This approach solves many of the problems inherent in procedural programming.

PROCEDURAL VS. OBJECT-ORIENTED

The object-oriented programming approach organizes programs in a way that mirrors the real world, in which all objects are associated with both attributes and activities. Using objects improves software reusability and makes programs easier to develop and easier to maintain. Programming in Python involves thinking in terms of objects; a Python program can be viewed as a collection of cooperating objects.

CHAPTER 12
INHERITANCE AND CLASS DESIGN

UNIVERSITEIT
GENT

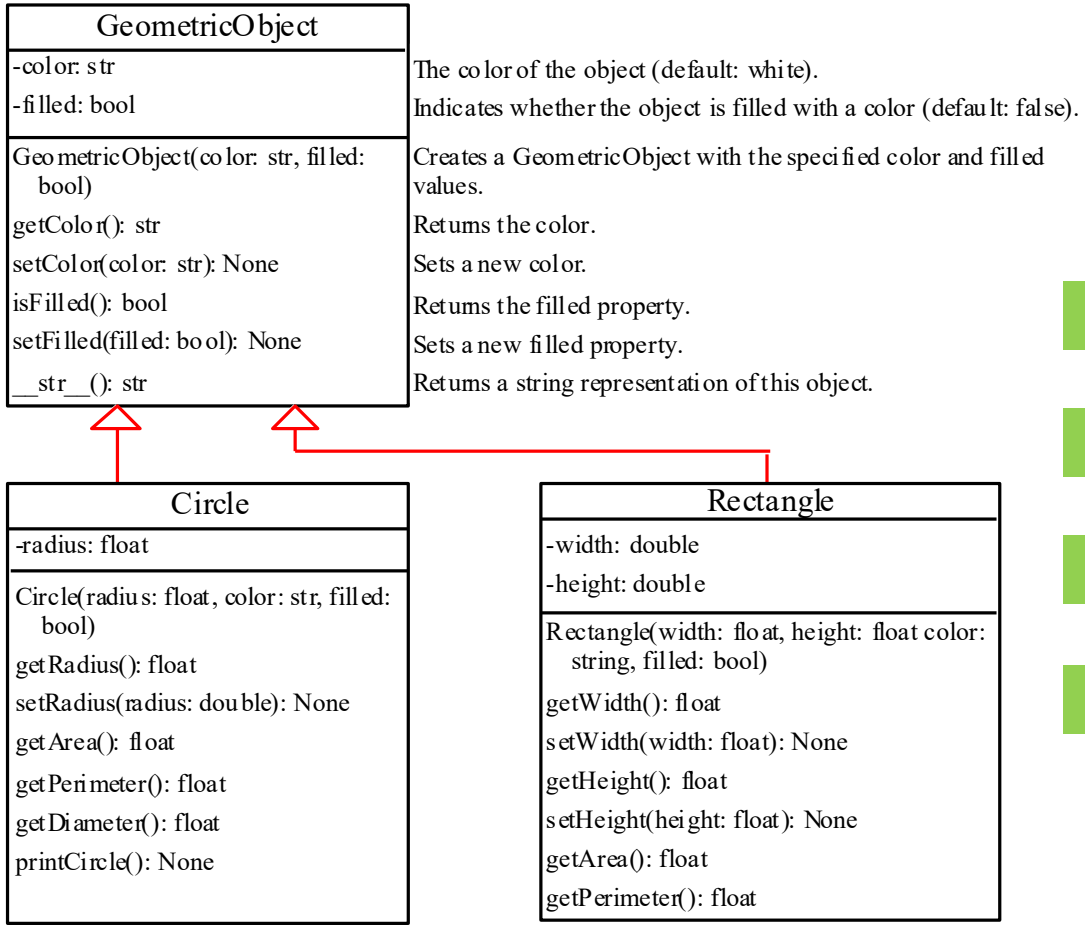
MOTIVATIONS

Suppose you will define classes to model circles, rectangles, and triangles. These classes have many common features. What is the best way to design these classes so to avoid redundancy? The answer is to use inheritance.

OBJECTIVES

- To develop a subclass from a superclass through inheritance (§12.2).
- To override methods in the subclass (§12.3).
- To explore the object class and its methods (§12.4).
- To understand polymorphism and dynamic binding (§12.5).
- To determine if an object is an instance of a class using the isinstance function (§12.6).
- To discover relationships among classes (§12.8).
- To design classes using composition and inheritance relationships (§§12.9-12.11).

SUPERCLASSES AND SUBCLASSES



GeometricObject

Circle

Rectangle

TestCircleRectangle

UNIVERSITEIT GENT

OVERRIDING METHODS

A subclass inherits methods from a superclass.

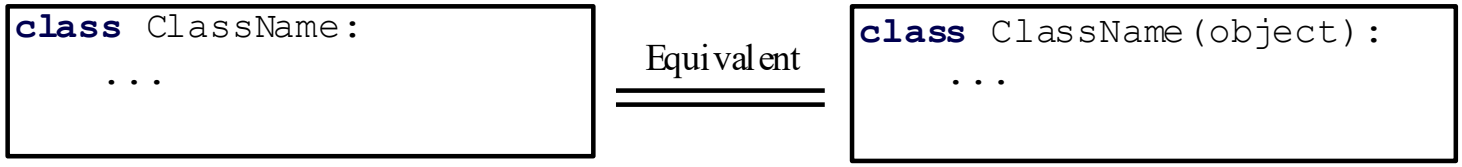
Sometimes it is necessary for the subclass to modify the implementation of a method defined in the superclass.

This is referred to as *method overriding*.

```
class Circle(GeometricObject):
    # Other methods are omitted
    # Override the __str__ method defined in GeometricObject
    def __str__(self):
        return super().__str__() + " radius: " + str(radius)
```

THE OBJECT CLASS

Every class in Python is descended from the object class. If no inheritance is specified when a class is defined, the superclass of the class is object by default.



There are more than a dozen methods defined in the object class. We discuss four methods __new__(), __init__(), __str__(), and __eq__(other) here.

THE __NEW__, __INIT__ METHODS

All methods defined in the object class are special methods with two leading underscores and two trailing underscores. The __new__() method is automatically invoked when an object is constructed. This method then invokes the __init__() method to initialize the object. Normally you should only override the __init__() method to initialize the data fields defined in the new class.

THE __STR__ METHOD

The __str__() method returns a string representation for the object. By default, it returns a string consisting of a class name of which the object is an instance and the object's memory address in hexadecimal.

THE __EQ__ METHOD

The __eq__(other) method returns True if two objects are the same. By default, x.__eq__(y) (i.e., x == y) returns False, but x.__eq__(x) is True. You can override this method to return True if two objects have the same contents.

POLYMORPHISM

- The three pillars of object-oriented programming are encapsulation, inheritance, and polymorphism.
- The inheritance relationship enables a subclass to inherit features from its superclass with additional new features. A subclass is a specialization of its superclass; every instance of a subclass is also an instance of its superclass, but not vice versa. For example, every circle is a geometric object, but not every geometric object is a circle. Therefore, you can always pass an instance of a subclass to a parameter of its superclass type.

PolymorphismDemo

DYNAMIC BINDING

Dynamic binding works as follows: Suppose an object o is an instance of classes C1, C2, ..., Cn-1, and Cn, where C1 is a subclass of C2, C2 is a subclass of C3, ..., and Cn-1 is a subclass of Cn. That is, Cn is the most general class, and C1 is the most specific class. In Python, Cn is the object class. If o invokes a method p, the JVM searches the implementation for the method p in C1, C2, ..., Cn-1 and Cn, in this order, until it is found. Once an implementation is found, the search stops and the first-found implementation is invoked.

UNIVERSITEIT GENT

THE ISINSTANCE FUNCTION

The isinstance function can be used to determine if an object is an instance of a class.

IsinstanceDemo

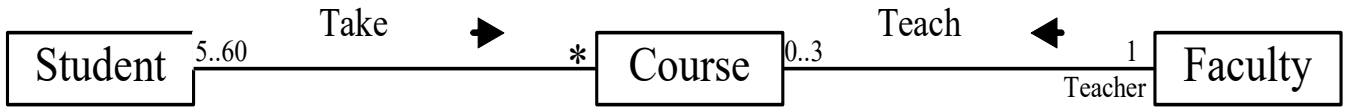
RELATIONSHIPS AMONG CLASSES

- Association
- Aggregation
- Composition
- Inheritance

UNIVERSITEIT
GENT

ASSOCIATION

- Association represents a general binary relationship that describes an activity between two classes.




```
class Student:
  def addCourse(self,
      course):
  # add course to a list
```

```
class Course:

  def addStudent(self,
      student):
    # add student to a list

  def setFaculty(self, faculty):
    # Code omitted
```

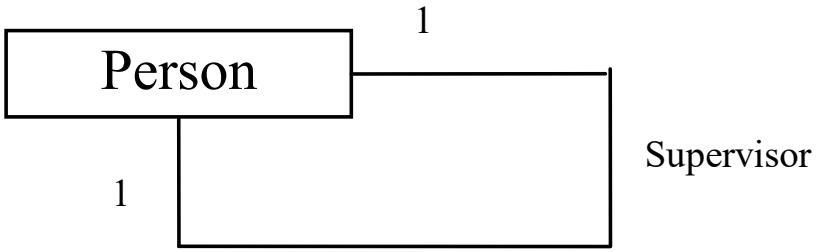
```
class Faculty:
  def addCourse(self,
     course):
     # add course to a list
```

An association is usually represented as a data field in the class.

UNIVERSITEIT GENT

ASSOCIATION BETWEEN SAME CLASS

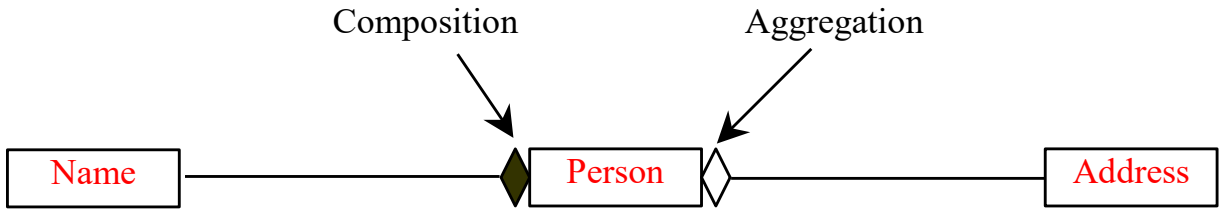
Association may exist between objects of the same class. For example, a person may have a supervisor.



UNIVERSITEIT GENT

AGGREGATION AND COMPOSITION

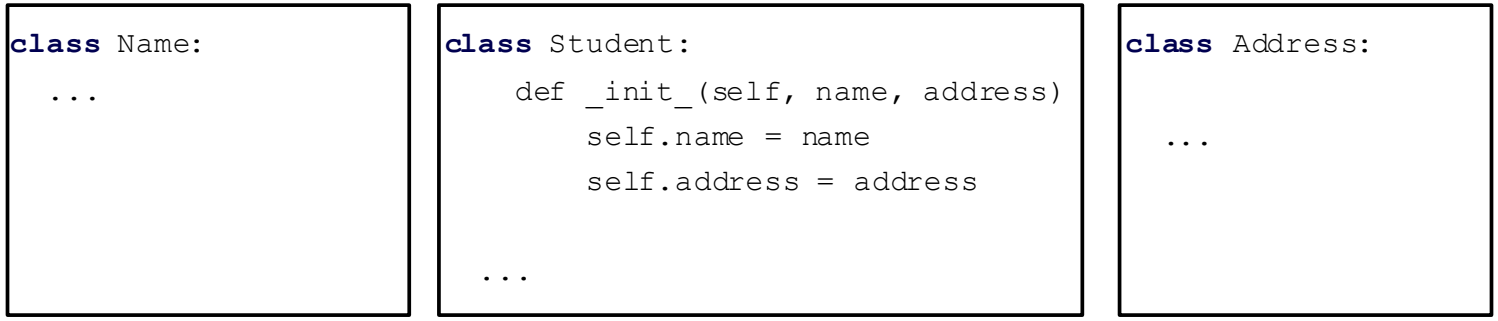
Aggregation is a special form of association, which represents an ownership relationship between two classes. Aggregation models the has-a relationship. If an object is exclusively owned by an aggregated object, the relationship between the object and its aggregated object is referred to as composition.



UNIVERSITEIT GENT

REPRESENTING AGGREGATION IN CLASSES

An aggregation relationship is usually represented as a data field in the aggregated class.



UNIVERSITEIT GENT

THE COURSE CLASS

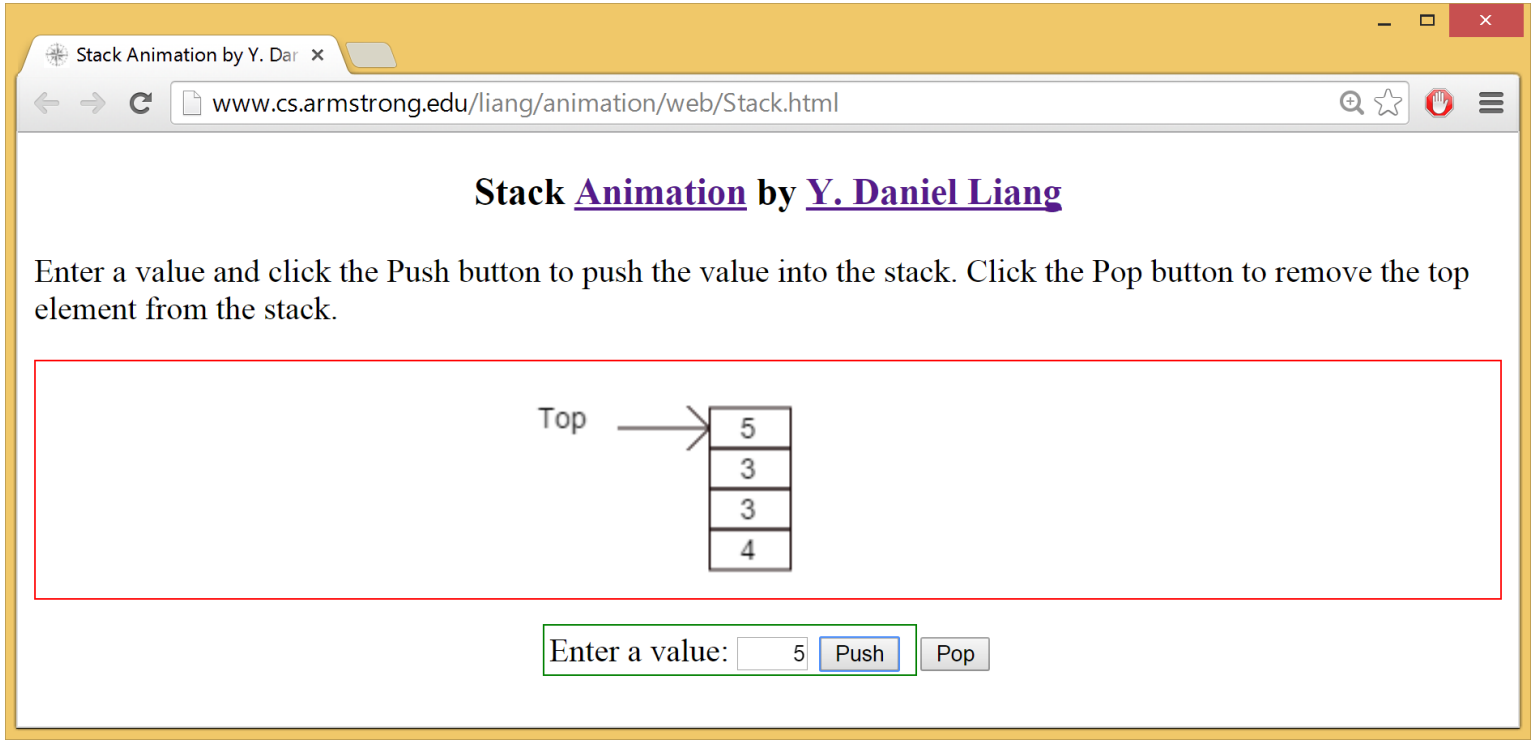
Course	
-courseName: str	The name of the course.
-students: list	An array to store the students for the course.
Course(courseName: str)	Creates a course with the specified name.
getCourseName(): str	Returns the course name.
addStudent(student: str): None	Adds a new student to the course.
dropStudent(student: str): None	Drops a student from the course.
getStudents(): list	Returns the students for the course.
getNumberOfStudents(): int	Returns the number of students for the course.

UNIVERSITEIT GENT

Course TestCourse

STACK ANIMATION

https://liveexample.pearsoncmg.com/dsanimation/StackeBook.html



THE STACK CLASS

- You can define a class to model stacks. You can use a list to store the elements in a stack. There are two ways to design the stack and queue classes:
- Using inheritance: You can define a stack class by extending list.
- Using composition: You can create a list as a data field in the stack class.

Both designs are fine, but using composition is better because it enables you to define a completely new stack class without inheriting the unnecessary and inappropriate methods from the list class.

UNIVERSITEIT GENT

THE STACK CLASS

Stack	
-elements: list	A list to store elements in the stack.
+Stack()	Constructs an empty stack.
+isEmpty(): bool	Returns True if the stack is empty.
+peek(): object	Returns the element at the top of the stack without removing it from the stack.
+push(value: object): None	Stores an element into the top of the stack.
+pop(): object	Removes the element at the top of the stack and returns it.
+getSize(): int	Returns the number of elements in the stack.

UNIVERSITEIT GENT

Stack TestStack

EXAMEN

- 90% van de eindscore
  - 45% MCE met hogere cesuur
  - 45% Dodona opdrachten
- Gesloten boek
- BYOD
- Extra oefeningen:
  - Boek
  - Dodona
  - Voorbeeldexamen via Dodona