ALGORITHMS DEVELOPING & PYTHON PROGRAMMING

By Dilan Hewage

First Print: 2021

Algorithms Developing & Python Programming

ISBN 978-624-98072-0-4

Other Books

Data Communication and Networking

Visit ictnotes.org for more information

Dear teachers and students,

I am an Advanced Level ICT teacher with more than seven years of experience. I graduated from the University of Colombo with a Bachelor of Information Technology degree. I am running my tech startup while teaching my beloved students. With the background and working with the latest technologies, I always try to give something beyond the syllabus.

I am pleased to inform you that I have written my second book for the Advanced Level Examination ICT subject. Getting a good result in ICT subjects is more complex as there is a gap between the paper and the syllabus. Therefore I tried to cover the missing gaps between the syllabus and examination paper to support students and teachers to overcome the challenge and be more thorough on the subject.

Most of us believe it is the most complicated section in the syllabus when it comes to programming. This myth becomes valid if you are trying to by-hard everything without understanding. Therefore to understand the subject you must practice by yourself. The Algorithm Developing and Python Programming book is written to give practical guidance to students with supporting material through our official website ictnotes.org and YouTube channel.

I advise you all to select the practicals and spend more time practicing the examples given in the book.

Best Regards,

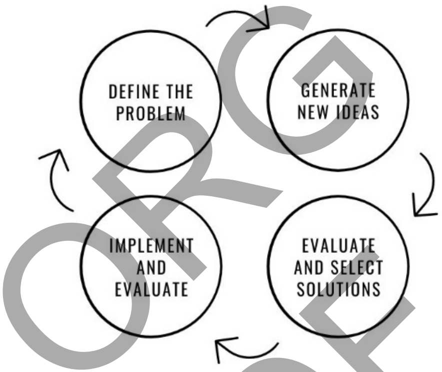
Dilan Sir

Note: Please scan the following QR code to the video guide to refer to the book and complete the practicals.

Table of Contents

1. PROBLEM-SOLVING PROCESS

Problem-solving is the act of defining a problem; determining the cause of the problem; identifying, prioritizing, and selecting alternatives for a solution; and implementing a solution.



1.1. Understanding the problem (Define the problem)

Diagnose the situation so that focus is on the problem, not just its symptoms. Helpful problem-solving techniques include using flowcharts to identify the expected steps of a process and cause-and-effect diagrams to define and analyze root causes.

The sections below help explain key problem-solving steps. These steps support the involvement of interested parties, factual information, comparison of expectations to reality, and a focus on the root causes of a problem.

It would be best if began by:

- Review and document how processes currently work (i.e., who does what, with what information, using what tools, communicating with organizations and individuals, in what time frame, using what format).
- Evaluate the possible impact of new tools and revised policies in developing the "what should be" model.

1.2. Defining the problem and boundaries

Postpone the selection of one solution until several problem-solving alternatives are identified. Considering multiple options can significantly enhance the value of your ideal solution. Once you have decided on the "what should be" model, this target standard becomes the basis for developing a road map for investigating alternatives. Brainstorming and team problem-solving techniques are both valuable tools in this stage of problem-solving.

Many alternative solutions to the problem should be generated before the final evaluation. In most cases first suggested solution is taken without considering the

3. PROGRAMMING PARADIGMS

3.1. Evolution of programming languages

Programming Language is indeed the fundamental unit of today's tech world. It is considered the set of commands and instructions that we give to the machines to perform a particular task.

For example, give some instructions to add two numbers. The machine will do it for us and tell us the correct answer accordingly. However, do you know that Programming Languages have a long and rich history of their evolution? Moreover, with a similar concern, here in this chapter, we will look at the development of Programming Languages over the period.

In the computer world, we have about 500+ programming languages that having their syntax and features. Moreover, suppose you type who is the father of the computer. In that case, the search engine will show you the result of Charles Babbage, but the father of the computer did not write the first code. It was Ada Lovelace who wrote the first-ever computer programming language, and the year was 1883.

1883: The Journey starts from here…!!

- In the early days, Charles Babbage had made the device. However, he was confused about how to give instructions to the machine. Then Ada Lovelace wrote the instructions for the analytical engine.
- Charles Babbage made the device, and Ada Lovelace wrote the code for computing Bernoulli's number.
- First time in history that the capability of computer devices was judged.

1949: Assembly Language

- It is a type of low-level language.
- It mainly consists of instructions (kind of symbols) that only machines could understand.
- In today's time, assembly language is also used in real-time programs such as simulation flight navigation systems and medical equipment, e.g., Fly-by-wire (FBW) systems.
- It is used to create computer viruses.

1952: Auto code

- Developed by Alick Glennie.
- The first compiled computer programming language
- COBOL and FORTRAN are the languages referred to as Autocode.

Figure 2: Alick Glennie

1957: FORTRAN

- Developers are John Backus and IBM.
- It was designed for numeric computation and scientific computing.
- Software for NASA probes voyager-1 (space probe) and voyager-2 (space probe) was initially written in FORTRAN 5.

1958: ALGOL

- ALGOL stands for **ALGO**rithmic **L**anguage.
- The initial phase of the most popular programming languages of C, C++, and JAVA
- It was also the first language implementing the nested function and has a simple syntax than FORTRAN.

1959: COBOL

- It stands for **Co**mmon **B**usiness-**O**riented **L**anguage.
- In 1997, 80% of the world's business ran on Cobol.
- The US internal revenue service scrambled its path to COBOL-based IMF (individual master file) in order to pay the tens of millions of payments mandated by the coronavirus aid, relief, and economic security.

1964: BASIC

- It stands for beginners All-purpose symbolic instruction code.
- In 1991 Microsoft released Visual Basic, an updated version of Basic
- The first microcomputer version of Basic was co-written by Bill Gates, Paul Allen, and Monte Davidoff for their newly-formed company, Microsoft.

Figure 3: Paul Allen (left) and Bill Gates in 1981, surrounded by some of the computers that ran their version of BASIC

4. PROGRAM TRANSLATORS

4.1. Need of program translation.

Modern programming languages attempt to give programmers the capability of doing complex things with a computer while writing instructions for the computer in a language close to their natural language. For example, most programmers are comfortable enough with the standard mathematical language to use expressions such as $\frac{1}{3}lwh$ (the volume of a pyramid with base length l, base width w, and height h). They do not care to know the sequence of machine instructions that are needed to evaluate this expression, much less the machine coding for those instructions.

Also, when a programmer wants to specify a numerical value, they prefer to specify it as a string of decimal digits rather than machine binary code. Furthermore, a programmer has no interest in the character coding used for the decimal digits. Thus, programming languages require powerful mechanisms for translating a programmer's language into a language that the machine understands.

4.2. Source program

Source program or source code is the original program written by the programmer. It is a text-based document. In the source program, the programmer writes the instructions the computer should perform. He writes these instructions using a computer programming language such as Java, C#.NET, etc. Programmers can easily understand and read the syntax of these programming languages. Furthermore, the written source code has to be according to the correct conventions and rules of that particular programming language.

4.3. Object program

Object Program or the object code is an executable machine file. The computer or the machine does not understand the source program or the source code. Therefore, the compiler converts the source program into an object program. In other words, the object program is the output of the compiler. It has instructions for the machine in the form of binary digits. Therefore, it is a machine-readable code. As the machine understands this object program, it is a machine-executable code. Additionally, suppose the programmer

5. INTEGRATED DEVELOPMENT ENVIRONMENT

5.1. Basic features of IDE.

IDEs provide interfaces for users to write code, organize text groups, and automate programming redundancies more fundamentally. However, instead of a bare-bones code editor, IDEs combine the functionality of multiple programming processes into one. Some IDEs focus on a specific programming language, such as Python or Java, but many have cross-language capabilities. IDEs often possess or allow the insertion of frameworks and element libraries to build upon base-level code in terms of text editing capabilities.

Throughout the writing process, users create hierarchies within the IDE and assign code groups to their designated region. From these, groupings can be strung together, compiled, and built. Most IDEs come with built-in debuggers, which activate upon the build. Visual debuggers are a substantial benefit of many IDEs. If any bugs or errors are spotted, users are shown which parts of the code have problems.

Key Benefits of Integrated Development Environments

- Serves as a single environment for most, if not all, of a developer's needs, such as version control systems, debugging tools, and Platform-as-a-Service.
- Code completion capabilities improve programming workflow.
- Automatically checks for errors to ensure top-quality code.
- Refactoring capabilities allow developers to make comprehensive and mistake-free renaming changes.
- Maintain a smooth development cycle.
- Increase developer efficiency and satisfaction.
- Deliver top-quality software on schedule.

5.2. IDE Common Features

Text editor

Virtually every IDE will have a text editor designed to write and manipulate source code. Some tools may have visual components to drag and drop front-end components, but most have a simple interface with language-specific syntax highlighting.

Debugger

Debugging tools assist users in identifying and remedying errors within source code. They often simulate real-world scenarios to test functionality and performance. Programmers and software engineers can usually test the various code segments and identify errors before the application is released.

Compiler

Compilers are components that translate programming language into a form machines can process, such as binary code. First, the machine code is analyzed to ensure its accuracy. The compiler then parses and optimizes the code to optimize performance.

Code completion

Code complete features assist programmers by intelligently identifying and inserting standard code components. These features save developers time writing code and reduce the likelihood of typos and bugs.

Programming language support

IDEs are typically specific to a single programming language, though several also offer multi-language support. As such, the first step is to figure out which languages you will be coding in and narrow your prospective IDE list down accordingly. Examples include Ruby, Python, and Java IDE tools.

Integrations and plugins

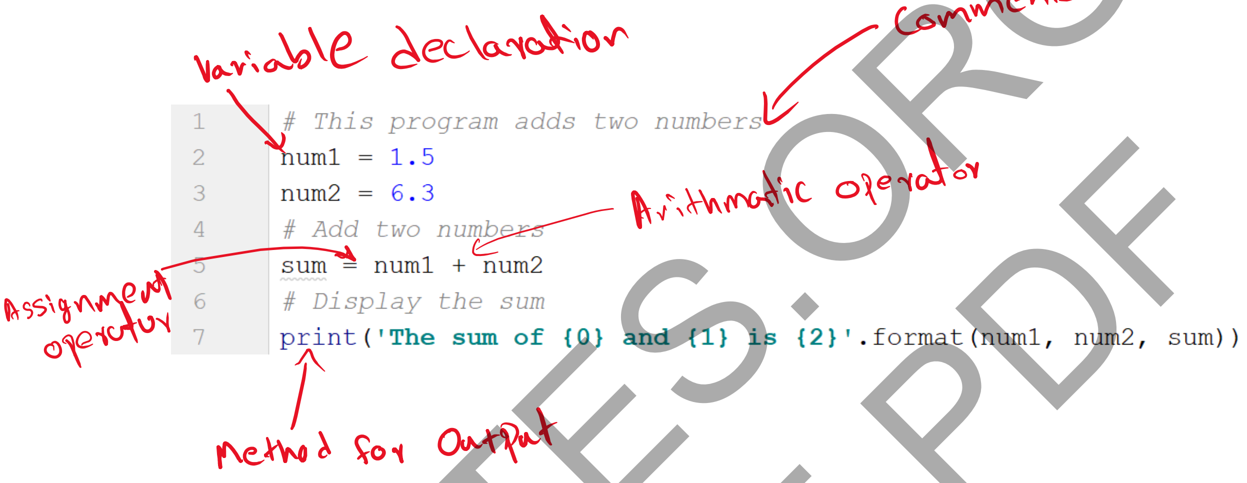
With the name integrated development environment, it is no surprise that integrations need to be considered when looking at IDEs. Your IDE is your development portal, so incorporating all your other development tools will improve development workflows and productivity. However, poor integrations can cause numerous issues and lead to many headaches, so make sure you understand how well a potential IDE fits into your ecosystem of existing tools.

Introduction to IDLE - Default Python IDE

IDLE stands for Integrated Development and Learning Environment. The story behind the name IDLE is similar to Python. Guido Van Rossum named Python after the British comedy group Monty Python while the name IDLE was chosen to pay tribute to Eric Idle, one of Monty Python's founding members. IDLE comes bundled with the default

6. USES AN IMPERATIVE PROGRAMMING LANGUAGE

6.1. Structure of a program



```
# This program adds two numbers
num1 = 1.5
num2 = 6.3
# Add two numbers
sum = num1 + num2
# Display the sum
print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))
```

6.2. Comments

A hash sign (#) that is not inside a string literal is the beginning of a comment. All characters after the #, up to the end of the physical line, are part of the comment and the Python interpreter ignores them.

Comments can be used,

- To explain Python code.
- To make the code more readable.
- To prevent execution when testing code.

A comment is a piece of text within a program that is not executed. It can be used to provide additional information to aid in understanding the code.

```
# This is the first comment
Print ("Hello, students!") # This is the second comment
```

The above code produces the following results –

```
Hello, students!
```

You can type a comment on the same line after a statement or expression –

```
name = "Kavindu" # This is another comment
```

Python does not have a multiple-line commenting feature. You have to comment on each line individually as follows –

```
# This is a comment.
# This is a comment, too.
# This is a comment, too.
```

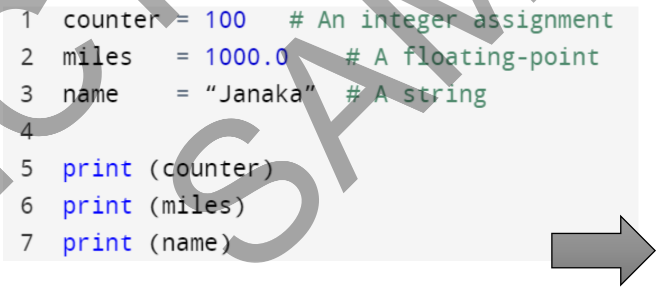
6.3. Constants and Variables

Variables are nothing but reserved memory locations to store values. It means that when you create a variable, you reserve some space in the memory. Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to the variables, you can store integers, decimals, or characters in these variables

Assigning Values to Variables

Python variables do not need an explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

The operand to the left of the "=" operator is the variable's name. The operand to the right of the "=" operator is the value stored in the variable.



```
counter = 100    # An integer assignment
miles   = 1000.0    # A floating-point
name    = “Janaka”  # A string

print (counter)
print (miles)
print (name)
```

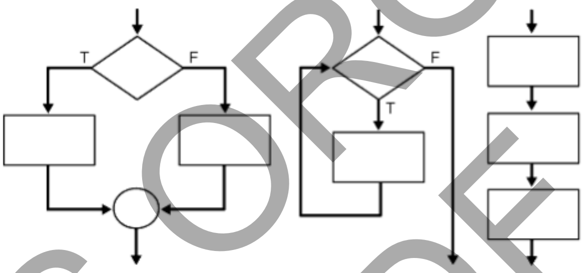
Output: Here, 100, 1000.0 and “Janaka” are the values assigned to counter, miles, and name variables, respectively. The code produces the following result

```
100
```

7. CONTROL STRUCTURES

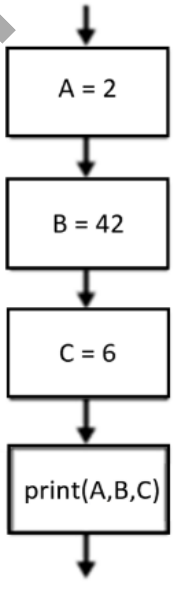
Control Structures are just a way to specify the flow of control in programs. Any algorithm or program can be more precise and understood if they use self-contained modules called logic or control structures. It analyzes and chooses in which direction a program flows based on specific parameters or conditions. There are three basic types of logic, or flow of control, known as:

1. Sequence logic, or sequential flow
2. Selection logic, or conditional flow
3. Iteration logic, or repetitive flow



7.1. Sequence

Sequential execution is when statements are executed one after another in order. You don't need to do anything more for this to happen. Sequential statements are a set of statements whose execution process occurs in a sequence. The problem with sequential statements is that if the logic has broken in any one of the lines, then the complete source code execution will break.



```
# This is a Squential statement
a = 20
b = 10
c = a - b
print(“Subtraction is : “, c)
```

7.2. Selection

Decision-making is anticipating conditions occurring while executing the program and specifying actions taken according to the conditions.

Decision structures evaluate multiple expressions which produce TRUE or FALSE as an outcome. You need to determine which action to take and which statements to execute if the outcome is TRUE or FALSE otherwise.

Following is the general form of a typical decision-making structure found in most of the programming languages –

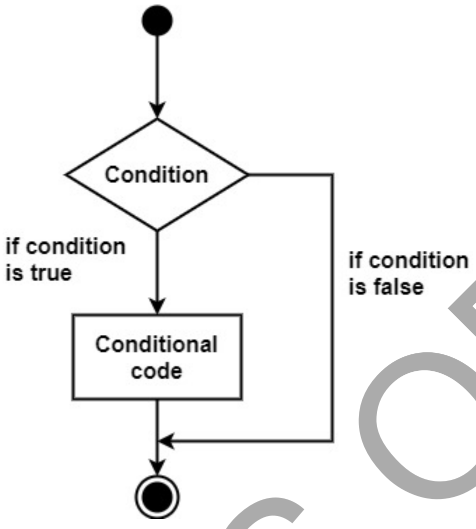


Figure 4: Selection control structure

Python supports the usual logical conditions from mathematics:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to a <= b
- Greater than: a > b
- Greater than or equal to a >= b

These conditions can be used in several ways, most commonly in "if statements" and loops. An "if statement" is written by using the if keyword.

If statement:

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

Note: Python relies on indentation (whitespace at the beginning of a line) to define the scope in the code. Other programming languages often use curly brackets for this purpose. If statement, without indentation (will raise an error):

```
a = 33
b = 200
```

9. USES DATA STRUCTURES IN PROGRAMS

9.1. Data structures

The basic Python data structures in Python include list, set, tuples, and dictionary. Each of the data structures is unique in its way. Data structures are “containers” that organize and group data according to type.

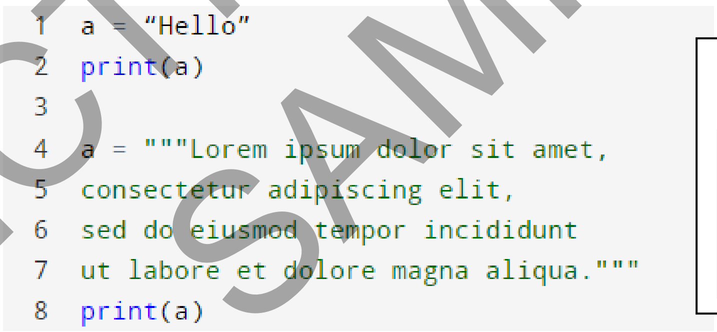
The data structures differ based on mutability and order. Mutability refers to the ability to change an object after its creation. Mutable objects can be modified, added, or deleted after they’ve been created, while immutable objects cannot be modified after their creation. Order, in this context, relates to whether the position of an element can be used to access the element.

9.2. Strings

Strings in Python are surrounded by either single quotation marks or double quotation marks.

‘hello’ is the same as “hello”.

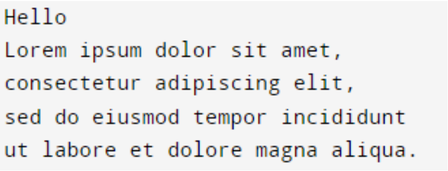
You can display a string literal with the print() function:



```
a = “Hello”
print(a)

a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

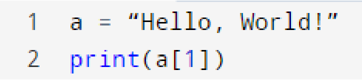
Output



```
Hello
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
```

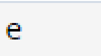
Strings are Arrays

Example: Get the character at position 1 (remember that the first character has the position 0)



```
a = “Hello, World!”
print(a[1])
```

Output

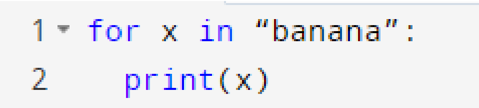


```
e
```

Looping Through a String

Since strings are arrays, we can loop through the characters in a string with a for a loop.

Loop through the letters in the word “banana”:



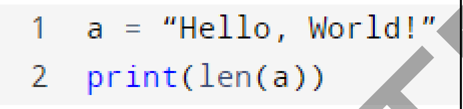
```
for x in “banana”:
    print(x)
```

```
Output

b
a
n
a
n
a
```

String Length

Example: The len() function returns the length of a string:



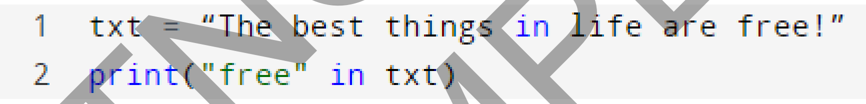
```
a = “Hello, World!”
print(len(a))
```

```
Output

13
```

Check String

Check if “free” is present in the following text:

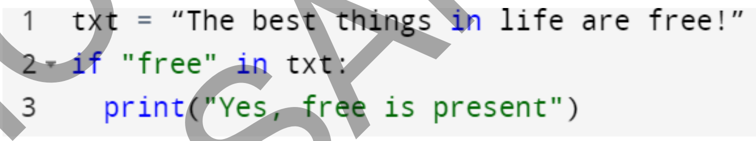


```
txt = “The best things in life are free!”
print("free" in txt)
```

```
Output

13
```

Example: Print only if “free” is present:



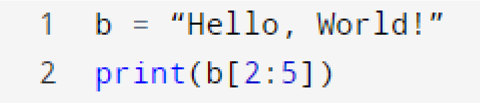
```
txt = “The best things in life are free!”
if "free" in txt:
  print("Yes, free is present")
```

```
Output

Yes, free is present
```

Slicing

You can return a range of characters by using the slice syntax. Specify the start index and the end index, separated by a colon, to return a part of the string. Example: Get the characters from position 2 to position 5 from “Hello, World!” string:



```
b = “Hello, World!”
print(b[2:5])
```

```
Output

llo
```

10. MANAGES DATA IN DATABASES

Note: Before you read this section, its is recommended to revise / learn My SQL basics, otherwise this chapter would not be clear for you. Here I've covered only basic Python

10.1. Connecting to a database

Install MySQL Driver

- Python needs a MySQL driver to access the MySQL database.
- We will use the driver "MySQL Connector".
- We recommend that you use PIP to install "MySQL Connector".
- PIP is most likely already installed in your Python environment.
- Navigate your command line to the location of PIP, and type the following:
- Download and install "MySQL Connector":
- C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>python -m pip install mysql-connector-python
- Now you have downloaded and installed a MySQL driver.

Test MySQL Connector

To test if the installation was successful, or if you already have "MySQL Connector" installed, create a Python page with the following content:

```
import mysql.connector
```

If the above code was executed with no errors, "MySQL Connector" is installed and ready to be used.

Create Connection

Start by creating a connection to the database. Use the username and password from your MySQL database:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)

print(mydb)
```

10.2. Retrieve data

Creating a Database

To create a database in MySQL, use the "CREATE DATABASE" statement:

Example: create a database named "mydatabase":

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)

mycursor = mydb.cursor()

mycursor.execute("CREATE DATABASE mydatabase")
```

Check if Database Exists

You can check if a database exist by listing all databases in your system by using the "SHOW DATABASES" statement:

Example: Return a list of your system's databases:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)

mycursor = mydb.cursor()

mycursor.execute("SHOW DATABASES")
```

11. SEARCHES AND SORTS DATA

11.1. Searching techniques

Searching for data stored in different data structures is a crucial part of pretty much every single application. There are many different algorithms available to utilize when searching. Each has other implementations and relies on various data structures to get the job done.

Choosing a specific algorithm for a given task is a crucial skill for developers. It can mean the difference between a fast, reliable and stable application and an application that crumbles from a simple request.

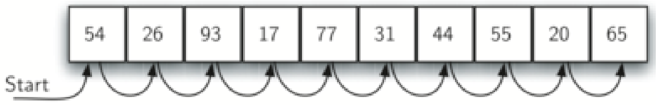
1. Membership Operators
2. Sequential / Linear Search
3. Binary Search
4. Jump Search
5. Fibonacci Search
6. Exponential Search
7. Interpolation Search

Here in this chapter, we are going to discuss only Sequential search only

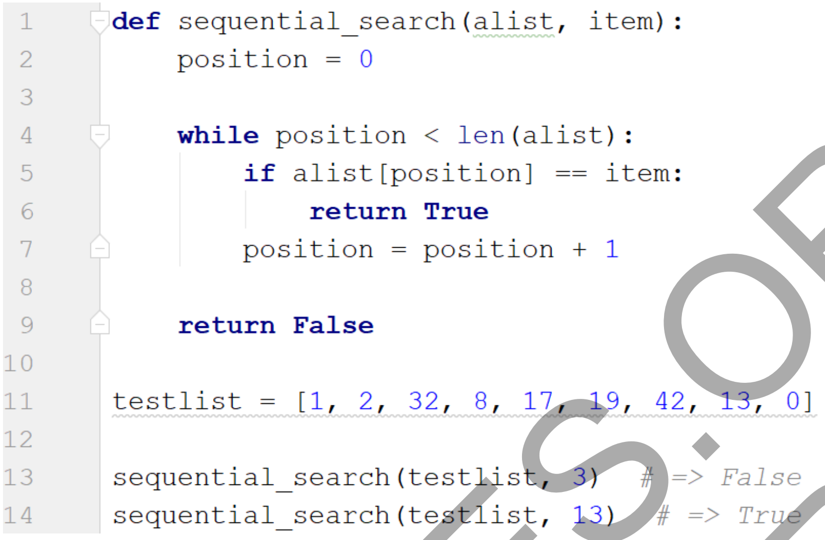
Sequential search

When data items are stored in a collection such as a list, we say that they have a linear or sequential relationship. Each data item is stored in a position relative to the others. In Python lists, these relative positions are the index values of the individual items. Since these index values are ordered, we can visit them in sequence. This process gives rise to our first searching technique, the sequential search.

The diagram below shows how this search works. Starting at the first item in the list, we simply move from item to item, following the underlying sequential ordering until we find what we are looking for or run out of items. If we run out of items, we have discovered that the item we were searching for was not present.



The Python implementation for this algorithm is shown below. The function needs the list and the item we are looking for and returns a boolean value as to whether it is present. Remember in practice, we would use the Python in operator for this purpose, so you can think of the below algorithm as what we would do if in were not provided for us.



```python
def sequential_search(alist, item):
    position = 0

    while position < len(alist):
        if alist[position] == item:
            return True
        position = position + 1

    return False

testlist = [1, 2, 32, 8, 17, 19, 42, 13, 0]

sequential_search(testlist, 3)   # => False
sequential_search(testlist, 13)   # => True
```

11.2. Sorting techniques

Sorting refers to arranging data in a particular format. Sorting algorithm specifies the way to arrange data in a particular order. Most common orders are in numerical or lexicographical order.

The importance of sorting lies in the fact that data searching can be optimized to a very high level, if data is stored in a sorted manner. Sorting is also used to represent data in more readable formats. Below we see five such implementations of sorting in python.

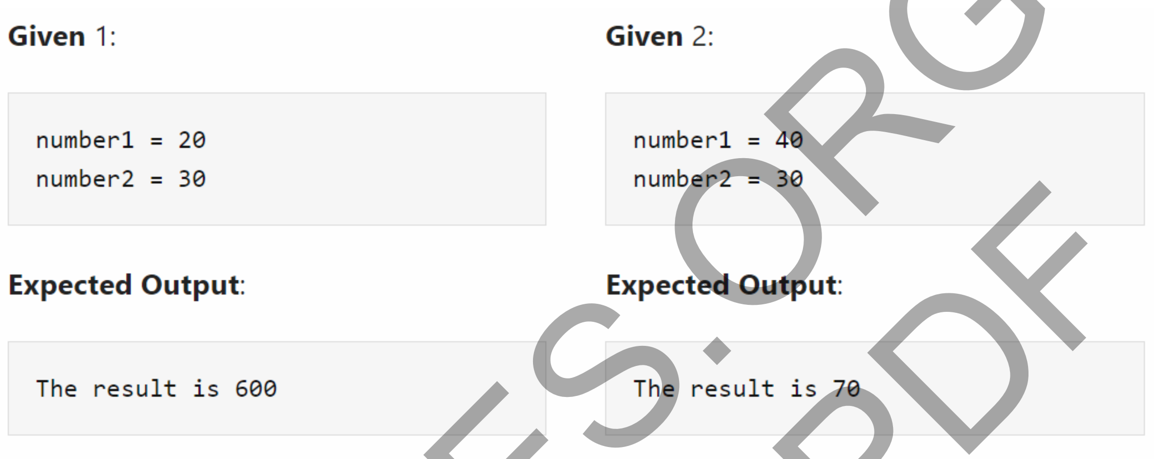
1. Bubble Sort
2. Merge Sort
3. Insertion Sort
4. Shell Sort
5. Selection Sort

Bubble sort

Bubble Sort is the simplest sorting algorithm that repeatedly swaps the adjacent elements if they are in the wrong order. It is a comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order.

13. MODEL QUESTIONS

Exercise 1: Given two integer numbers, return their product. If the product is greater than 1000, then return their sum



Given 1:

```
number1 = 20
number2 = 30
```

Expected Output:

```
The result is 600
```

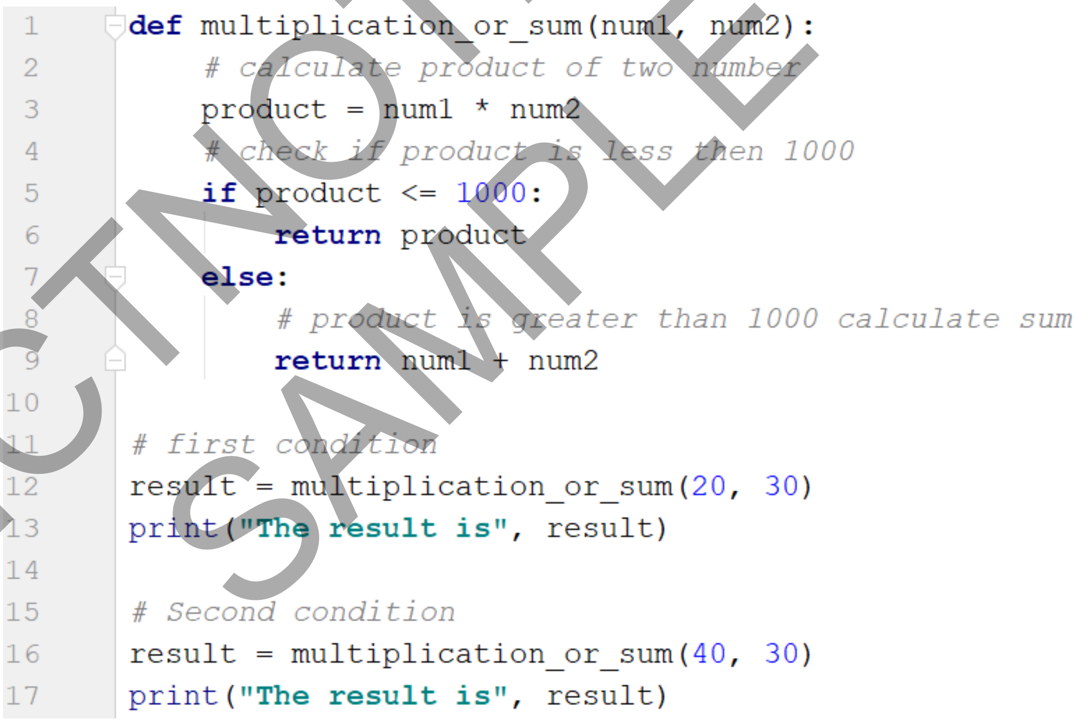
Given 2:

```
number1 = 40
number2 = 30
```

Expected Output:

```
The result is 70
```

Solution:



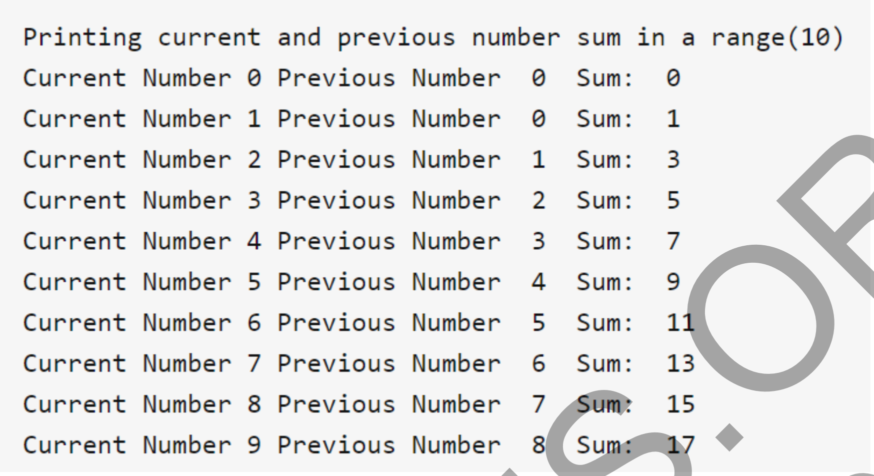
```python
def multiplication_or_sum(num1, num2):
    # calculate product of two number
    product = num1 * num2
    # check if product is less then 1000
    if product <= 1000:
        return product
    else:
        # product is greater than 1000 calculate sum
        return num1 + num2

# first condition
result = multiplication_or_sum(20, 30)
print("The result is", result)

# Second condition
result = multiplication_or_sum(40, 30)
print("The result is", result)
```

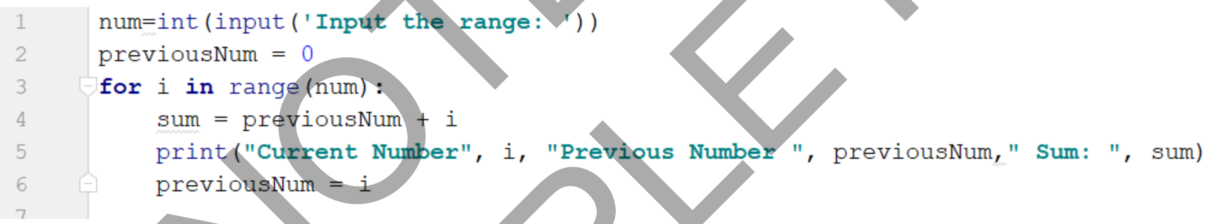
Exercise 2: Given a range of the first 10 numbers, Iterate from the start number to the end number, and In each iteration, print the sum of the current number and previous number

Expected Output



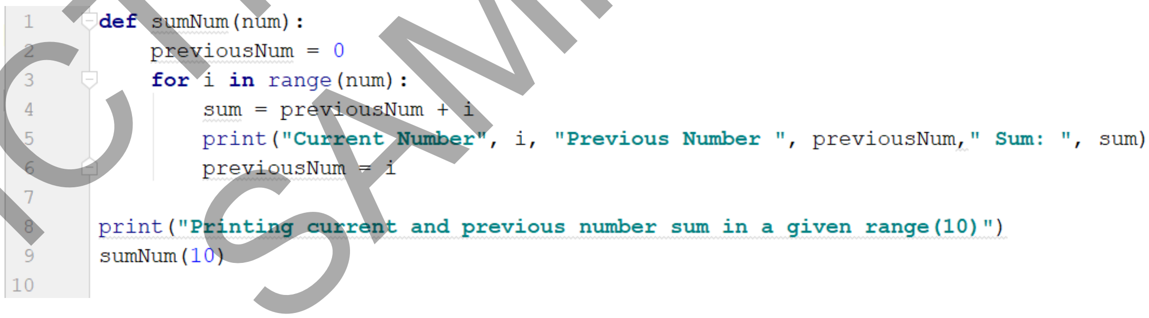
```
Printing current and previous number sum in a range(10)
Current Number 0 Previous Number  0  Sum:  0
Current Number 1 Previous Number  0  Sum:  1
Current Number 2 Previous Number  1  Sum:  3
Current Number 3 Previous Number  2  Sum:  5
Current Number 4 Previous Number  3  Sum:  7
Current Number 5 Previous Number  4  Sum:  9
Current Number 6 Previous Number  5  Sum:  11
Current Number 7 Previous Number  6  Sum:  13
Current Number 8 Previous Number  7  Sum:  15
Current Number 9 Previous Number  8  Sum:  17
```

Solution 1 without function



```python
num=int(input('Input the range: '))
previousNum = 0
for i in range(num):
    sum = previousNum + i
    print("Current Number", i, "Previous Number ", previousNum," Sum: ", sum)
    previousNum = i
```

Solution 2 with function



```python
def sumNum(num):
    previousNum = 0
    for i in range(num):
        sum = previousNum + i
        print("Current Number", i, "Previous Number ", previousNum," Sum: ", sum)
        previousNum = i

print("Printing current and previous number sum in a given range(10)")
sumNum(10)
```

ABd1234@1

Hints:

In case of input data being supplied to the question, it should be assumed to be a console input.

Solutions:

```
import re
value = []
items=[x for x in input().split(',')]
for p in items:
    if len(p)<6 or len(p)>12:
        continue
    else:
        pass
    if not re.search("[a-z]",p):
        continue
    elif not re.search("[0-9]",p):
        continue
    elif not re.search("[A-Z]",p):
        continue
    elif not re.search("[$#@]",p):
        continue
    elif re.search("\s",p):
        continue
    else:
        pass
    value.append(p)
print (",".join(value))
```

Exercise 26: Define a class with a generator that can iterate the numbers, which are divisible by 7, between a given range 0 and n.