Introduction to Python Programming

OBJECTIVES:

- To read and write simple Python programs.
- To develop Python programs with conditionals and loops.
- To def ine Python functions and call them.
- To use Python data structures –- lists, tuples, dictionaries.
- To do input/output with f les in Python.

Course Outline

- Lecture 1: Introduction to Python
- Lecture 2: Control Structures
- Lecture 3: Functions
- Lecture 4: Data Structures
- Lecture 5: Object-Oriented Programming
- Lecture 6: File I/O and Modules

Lecture 1: Introduction to Python

- What is Python?
- Why Python is popular?
- How to install Python
- Running Python from the command line
- Writing and running your f irst Python program
- Basic data types in Python: integers, f bats, strings, and booleans
- Basic operations on data types: arithmetic, comparison, and logical operators
- Variables and assignment statements

Lecture 2: Control Structures

- Conditional statements: if, else, elif
- Boolean operators: and, or, not
- Loops: for and while
- Iteration over data structures: lists, tuples, and dictionaries
- Loop control statements: break, continue, and pass

Lecture 3: Functions

- Def ining and calling functions
- Function arguments and return values
- Function scope and variable scope
- Lambda functions and anonymous functions
- Recursion and iterative algorithms

Lecture 4: Data Structures

- Lists: indexing, slicing, appending, and sorting
- Tuples: immutable sequences
- Dictionaries: key-value mappings
- Sets: unordered collections of unique elements
- List comprehensions and generator expressions

Lecture 5: Object-Oriented Programming

- Objects and classes: creating and using classes
- Class attributes and instance attributes
- Constructors and destructors
- Inheritance and polymorphism
- Abstract classes and interfaces

Lecture 6: File I/O and Modules

- Reading from and writing to files
- Exception handling: try, except, else, and finally
- Creating and using modules

What is Python?

- Python is a high-level, interpreted programming language that was first released in 1991 by Guido van Rossum.
- It was further developed by the Python Software Foundation.
- It is a general-purpose language that can be used for web development, scientific computing, data analysis, artificial intelligence, and much more.
- Python is known for its simplicity, readability, and ease of use.

- Its syntax is straightforward and intuitive, making it a great language for beginners.
- At the same time, Python is powerful enough to handle complex programming tasks, making it a popular choice among experienced developers as well.

Why Python is popular?

- Readability: Python code is easy to read and understand, thanks to its simple and consistent syntax.
- Flexibility: Python can be used for a wide range of applications, from web development to scientif ic computing to machine learning.
- Rich ecosystem: Python has a vast array of third-party libraries and tools that make it easy to accomplish a wide range of tasks.
- Open source: Python is open source software, which means it's free to use and distribute.

How to install Python

- To start using Python, you first need to install it on your computer.
- Python is available for Windows, macOS, and Linux, and you can download the latest version from the official Python website (https://www.python.org/downloads/).
-

- Once you have downloaded the installer, run it and follow the instructions to install Python on your computer.
- After installing Python, you can open a command prompt or terminal window and type "python" to start the Python interpreter.
- This will open up an interactive shell where you can type Python code and see the results immediately.

Writing and running your first Python program

- Let's write and run our first Python program.
- Open up a text editor (such as Notepad or Sublime Text) and type the following code:
- print("Hello, world!")
- Save this file as "hello.py" in a convenient location.

- Open a command prompt or terminal and navigate to the directory where you saved the f le.
- Then, run the program by typing python hello.py and pressing enter.
- Alternatively, you can install Integrated Development Environment (IDE) such pycharm, spyder or jupiter notebook.

What is Indentation in Python

- Whitespace is used for indentation in Python.
- Unlike many other programming languages which only serve to make the code easier to read, Python indentation is mandatory.
- Block can be regarded as the grouping of statements for a specif ic purpose.

- Most programming languages like C, C++, and Java use braces { } to def ne a block of code for indentation.
- One of the distinctive roles of Python is its use of indentation to highlight the blocks of code.
- If a block has to be more deeply nested, it is simply indented further to the right.

Basic data types in Python

- Integers: whole numbers, such as 1, 2, 3, and so on.
- Floats: numbers with a decimal point, such as 3.14 or 2.71828.
- Strings: sequences of characters, enclosed in quotes (single or double).
- Booleans: either True or False.

Basic operations on data types

- Arithmetic operators: +, -, *, /, //, %, **
- Comparison operators: ==, !=, <, >, <=, >=
- Logical operators: and, or, not

Variables and assignment statements

- In Python, we can store values in variables using assignment statements. For example:
- x = 10
- y = 3.14
- name = "Alice"
- is_python_fun = True
-

- We can then use these variables in our code, perform operations on them, and assign new values to them.
- print(x + y) # output: 8.14
- print("Hello, " + name) # output: Hello, Alice
- print(not is_true) # output: False
-

Control structures

- Control structures are statements that allow us to control the f ɒw of execution in our programs.
- In Python, there are three main types of control structures: conditional statements, loops, and functions.
-
-

Conditional statements

- In Python programming language, the type of control f ɒw statements are as follows:
- The if statement
- The if-else statement
- The nested-if statement
- The if-elif-else ladder
-

if statement

- The if statement is the most simple decision-making statement.
- It is used to decide whether a certain statement or block of statements will be executed or not.
- Syntax:

- if condition:
- # Statements to execute if
- For example:
- x = 10
- if x > 0:
- print("x is positive")
-

if-else statement

- The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won’t.
- we can use the else statement with if statement to execute a block of code when the if condition is false.

- if (condition):
- # Executes this block if
- # condition is true
- else:
- # Executes this block if
- # condition is false

```
i = 20
if (i < 15):
    print("i is smaller than 15")
    print("i'm in if Block")
else:
    print("i is greater than 15")
    print("i'm in else Block")
print("i'm not in if and not in else Block")
```

nested-if statement

- A nested if is an if statement that is the target of another if statement.
- Nested if statements mean an if statement inside another if statement.

- Syntax:
- if (condition1):
- # Executes when condition1 is true
- if (condition2):
- # Executes when condition2 is true
- # if Block is end here
- # if Block is end here

```
i = 10
if (i == 10):
    #  First if statement
    if (i < 15):
        print("i is smaller than 15")
    # Nested - if statement
    # Will only be executed if statement above
    # it is true
    if (i < 12):
        print("i is smaller than 12 too")
    else:
        print("i is greater than 15")
```

if-elif-else ladder

- As soon as one of the conditions controlling the if is true, the statement associated with that if is executed.
- The rest of the ladder is bypassed.
- If none of the conditions is true, then the f inal else statement will be executed.

- Syntax:
- if (condition):
- statement
- elif (condition):
- statement
- .
- .
- else:
- statement

```
x = 10
if x > 0:
    print("x is positive")
elif x < 0:
    print("x is negative")
else:
    print("x is zero")
```

- In this example, we f irst check if x is greater than 0. If it is, we print "x is positive".
- If it is not, we check if x is less than 0. if it is, we print "x is negative".
- If neither of these conditions is true (i.e., x is equal to 0), we print "x is zero".
-

Loops

- Iteration refers to repeating the same thing over and over again.
- The iteration will only terminate when a certain condition is met.
- There are two types of loops in python:
- For loop and while loop

For loop

- For loops iterate over a given sequence.
- These sequences can either be a list, tuple or string.
- It can also iterate over a range of intergers.

- for < variable > in < sequence >:
    - < body of for loop >
- < s t a t e m e n t s after for loop >
- In this code the <variable> is any variable name you choose.
- The variable will be
- assigned to the f ṙst element of the sequence and then the statements in the body of the for loop will be executed.

- Then, the variable is assigned to the second element of the sequence and the body of the for loop is repeated.
- This continues until no elements are left in the sequence.

- For loops can iterate over a sequence of numbers using the "range" function.
- # Prints out the numbers 0,1,2,3,4
- for x in range(5):
- print(x)
-
- # Prints out 3,4,5
- for x in range(3, 6):
- print(x)
-
- # Prints out 3,5,7
- for x in range(3, 8, 2):
- print(x)

- For loops iterate over a given sequence.
- primes = [2, 3, 5, 7]
- for prime in primes:
- print(prime)

While Loop

- The while loop exists in almost all programming languages and is used to iterate (or repeat) one or more code statements as long as the test condition (expression) is True.
- This iteration construct is usually used when the number of times we need to repeat the block of code to execute is not known.

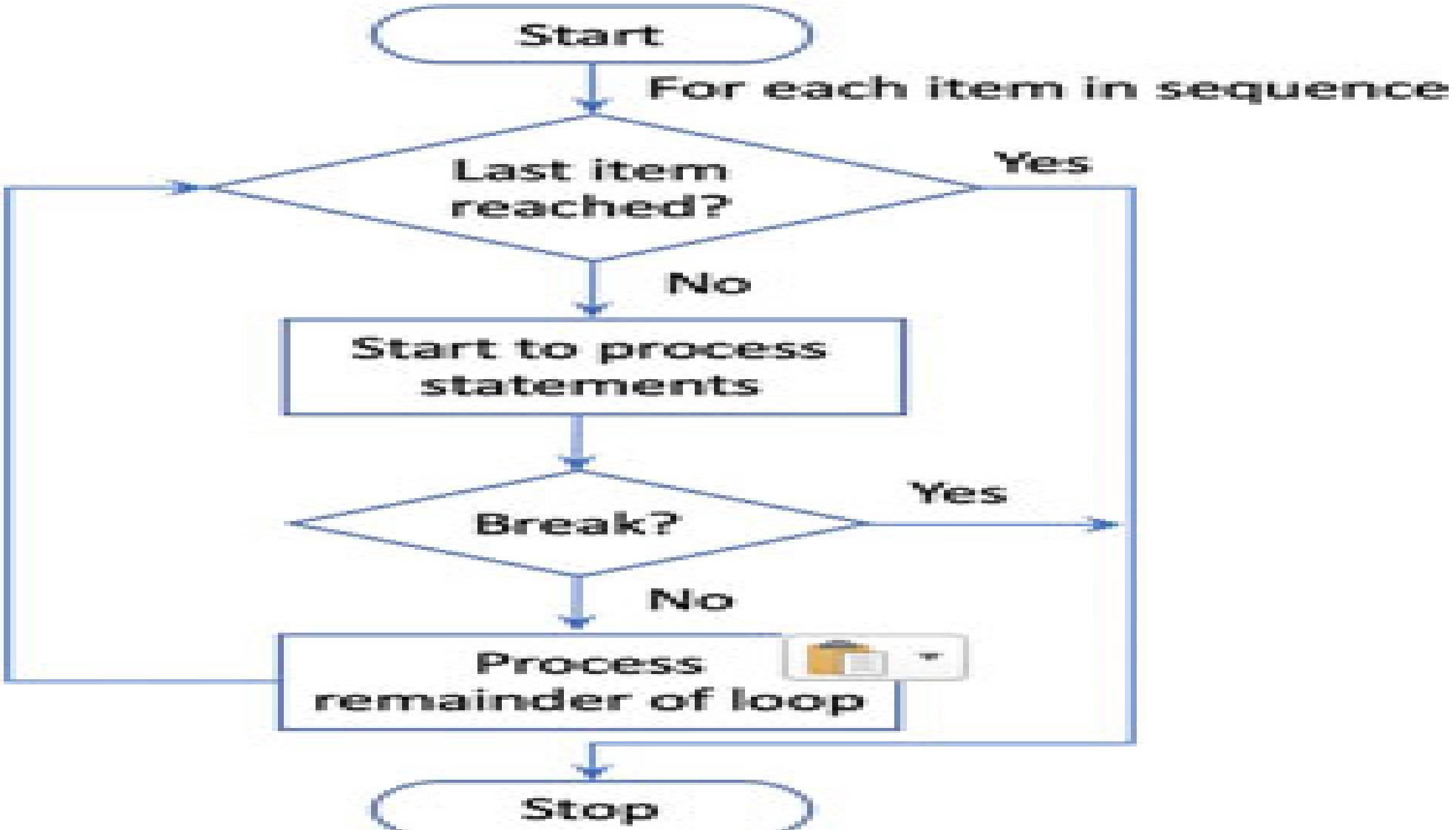
- For example, it may need to repeat until some solution is found or the user enters a particular value.
- The Python while loop has the basic form:
- while <test-condition-is-true>:
  - statement or statements

- count = 0
- print('Starting')
- while count < 10:
  - print(count, ' ', end='') # part of the while loop
  - count += 1
  - # also part of the while loop
- print() # not part of the while loop
- print('Done')

Break Loop Statement

- Python allows programmers to decide whether they want to break out of a loop early or not (whether we are using a for loop or a while loop).
- This is done using the break statement.

The break statement, when executed, will terminate the current loop and jump the program to the first line after the loop.


Start
For each item in sequence
Last item reached?
Yes
No
Start to process statements
Break?
Yes
No
Process remainder of loop
Stop

- Typically, a guard statement (if statement) is placed on the break so that the break statement is conditionally applied when appropriate.
- print('Only print code if all iterations completed')
- num = int(input('Enter a number to check for: '))
- for i in range(0, 6):
  - if i == num:
    - break
  - print(i, ' ', end='')
- print('Done')

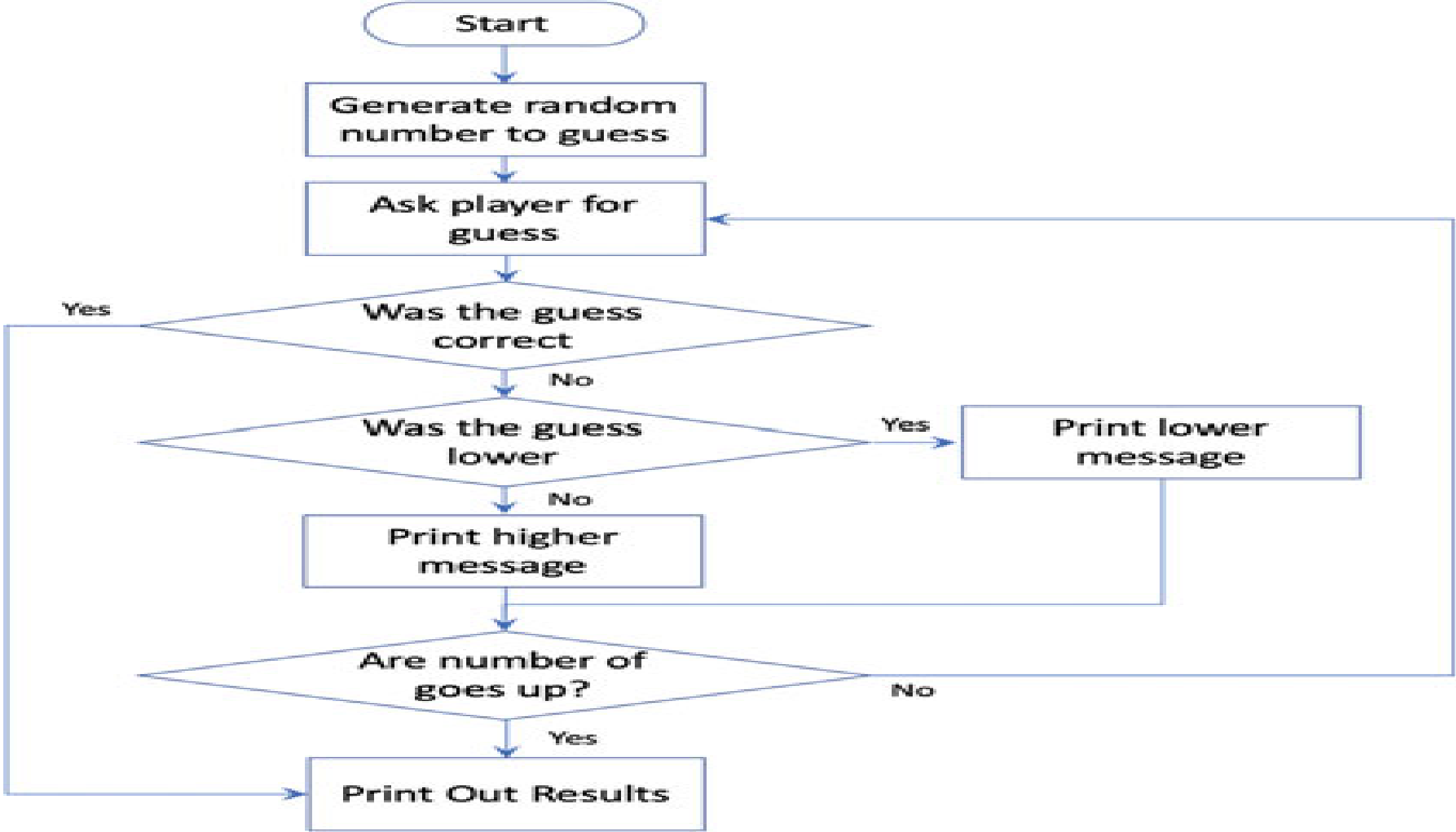
- The continue statement also affects the f ow of control within the looping constructs for and while.
- However, it does not terminate the whole loop; rather it
- only terminates the current iteration round the loop.
- This allows you to skip over part of a loop’s iteration for a particular value,
- but then to continue with the remaining values in the sequence.

Number Guessing Game

- we are going to bring everything we have learned so far together to create a simple number guessing game.
- This will involve creating a new Python program, handling user input, using the
- if statement as well as using looping constructs.
- We will also, use an additional library or module, that is not by default available
- by default to your program; this will be the random number generator module.

What Will the Program Do?

- The aim of our number guess game is to guess the number that the program has come up with.


Start
Generate random number to guess
Ask player for guess
Was the guess correct
Yes
No
Was the guess lower
Yes
Print lower message
No
Print higher message
Are number of goes up?
No
Yes
Print Out Results

- The program randomly selects a number between 1 and 10.
- It will then ask the player to enter their guess.
- It will then check to see if that number is the same as the one the computer
- randomly generated; if it is then the player has won.
- If the player’s guess is not the same, then it will check to see if the number is

- higher or lower than the guess and tell the player.
- The player will have 4 goes to guess the number correctly; if they don’t guess
- the number within this number of attempts, then they will be informed that they
- have lost the game and will be told what the actual number was.

List

- In Python, a list is a collection of items that are ordered and changeable.
- It is one of the most commonly used data structures in Python.
- the value in a list can be any type. The values in a list are called elements or sometimes items.

• Creating a List

- There are several ways to create a new list; the simplest is to enclose the
- elements in square brackets ([and]): [10, 20, 30, 40] ['crunchy frog', 'ram bladder', 'lark vomit'].
- The f irst example is a list of four integers. The second is a list of three strings.
- The elements of a list don’t have to be the same type.
- The following list contains a string, a f bat, an integer, and (lo!) another list: ['spam', 2.0, 5, [10, 20]]

Access Items

- You can access individual elements of a list by using indexing.
- Indexing in Python starts from 0, so the f irst element of the list has an index of 0, the second element has an index of 1, and so on.
- For example, to access the f irst element of my_list, you would use my_list[0].
- -1 refers to the last item, -2 refers to the second last item etc.

- thislist = ["apple", "banana", "cherry"]
- print(thislist[-1])

Range of Indexes

- You can specify a range of indexes by specifying where to start and where to end the range.
- When specifying a range, the return value will be a new list with the specified items.
-
- thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
- print(thislist[2:5])
- thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
- print(thislist[:4])
- thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
- print(thislist[2:])
-

Change Item Value

- thislist = ["apple", "banana", "cherry"]
- thislist[1] = "blackcurrant"
- print(thislist)
- thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
- thislist[1:3] = ["blackcurrant", "watermelon"]
- print(thislist)

Append and Remove Items

- To add an item to the end of the list, use the append() method:
- thislist = ["apple", "banana", "cherry"]
- thislist.append("orange")
- print(thislist)
- To insert a list item at a specif ied index, use the insert() method.
- thislist = ["apple", "banana", "cherry"]
- thislist.insert(1, "orange")
- print(thislist)

- The remove() method removes the specified item
- thislist = ["apple", "banana", "cherry"]
- thislist.remove("banana")
- print(thislist)
- The pop() method removes the specified index.
- thislist = ["apple", "banana", "cherry"]
- thislist.pop(1)
- print(thislist)

maintained separately.

One way in which these units can be defined is as Python functions.

Functions

In Python functions are groups of related statements that can be called together,

that typically perform a specific task, and which may or may not take a set of parameters

or return a value.

It also means that the same function can be called multiple times or in multiple locations.
This help to ensure that although a piece of functionality is used in multiple places.
When a function is called (or invoked) the flow of control a program jumps from where the function was called to the point where the function was defined.
The body of the function is then executed before control returns back to where it was called from.

Types of Functions

Technically speaking there are two types of functions in Python; built-in functions and user-defined functions.

Built-in functions are those provided by the language and we have seen several of these already. For example, both print() and input() are built-in functions. We did not need to define them ourselves as they are provided by Python.

In contrast user-defined functions are those written by developers.

Defining functions

The basic syntax of a function is illustrated below:

```
def function_name(parameter list):
                    statement
                    statement(s)
```

1. All (named) functions are defined using the keyword def; this indicates the start of a function definition. A keyword is part of the syntax of the Python language and cannot be redefined and is not a function.
2. A function can have a name which uniquely identifies it; you can also have anonymous functions, but we will leave those until later in this chapter.
3. The naming conventions that we have been adopting for variables also applies to functions, they are all lower case with the different elements of the function name separated by an '_'.

4. A function can (optionally) have a list of parameters which allow data to be passed into the function. These are optional as not all functions need to be supplied with parameters.
5. A colon is used to mark the end of the function header and the start of the function body. The function header defines the signature of the function (what its called and the parameters it takes). The function body defines what the function does.
6. An optional documentation string (the docstring) can be provided that describes what the function does. We typically use the triple double quote string format as this allows the documentation string to go over multiple lines if required.

relative to the function definition. All lines that are indented are part of the function until a line which is intended at the same level as the def line.
8. It is common to use 4 spaces (not a tab) to determine how much to indent the body of a function by.

```
def print_msg():
    print('Hello World!')
```

An Example Function

This function is called print_msg and when called (also known as invoked) it will run the body of the function which will print out the string, for example

```
print_msg()
```

```
Hello World!
```

```
def print_my_msg(msg):
print(msg)

print_my_msg('Hello World')
print_my_msg('Good day')
print_my_msg('Welcome')
```

Whenever a return statement is encountered within a function then that function will terminate and return any values following the return keyword

This means that if a value is provided, then it will be made available to any calling code.

Returning Values from Functions

For example, the following defines a simple function that squares whatever value has been passed to it:

```
def square(n):
return n * n
```

```
def swap(a, b):
    return b, a
```