Chapter 1 Introduction to Python

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code. Experienced programmers in any programming language can pick up Python very quickly. It is also easy for beginners to learn.

1. Installing Python IDE

First, you need to install a Python IDE, which is a software that allows you to code and run your Python programs on your own computer. You can use any IDE such as the ones listed below:

Thonny

- Download: https://thonny.org/

VS Code:

- Download: https://code.visualstudio.com/download
- How to Install and Configure VS Code on Windows: https://www.youtube.com/watch?v=cUAK4x_7thA

PyCharm

- Download PyCharm Community Edition: https://www.jetbrains.com/pycharm/download/#section=windows

- How to Install and Configure PyCharm on Windows: https://www.liquidweb.com/kb/how-to-install-and-configure-pycharm-on-windows/

Example 01: A python program that displays a message "Welcome to Python"

```
print("Welcome to Python")
```

Output

```
Welcome to Python
```

The `print()` function is used to output the specified message to the console window.

2. Basics of Python

2.1 Comments

In Python, there are two type of comments:

- **Comment line** or **Single-line comment** – a comment that is preceded by a pound sign (#).
- **Paragraph comment** or **Multi-line comment** – a comment that is enclosed between three consecutive single quotation marks (''') on one or several lines.

Example 02: Using line comments.

```
# Displays two messages
print("Welcome to Python.") # displays Welcome to Python.
print("Python is fun.") # displays Python is fun.
```

Output:

```
Welcome to Python.
Python is fun.
```

Example 03: Using a paragraph comment.

```
'''
This program displays Welcome to Python and
Python is fun
'''
print("Welcome to Python.")
print("Python is fun.")
```

Output:

```
Welcome to Python.
Python is fun.
```

2.2 Identifiers Rules

Identifiers are the names that identify the elements such as variables and functions in a program. As you can see in the previous example, print is the name of an element that appear in the program. In programming terminology, such names are called identifiers. All identifiers must obey the following rules:

- a sequence of characters that consists of letters, digits, and underscores (_).
- must start with a letter or an underscore. It cannot start with a digit.
- cannot be a keyword.
- can be of any length.

Python Keywords

Keywords have special meanings in Python. They should not be used for anything other than their predefined purpose in Python.

and	else	in	return
as	except	is	True
assert	False	lambda	try
break	finally	None	while
class	for	nonlocal	with
continue	from	not	yield
def	global	or	
del	if	pass	
elif	import	raise	

2.3 Using Variables

Unlike other programming languages; Python does not require you to declare variables.

Example 04: Using Variables

```
x = 1
y = 2

print(x + y)
```

Output

```
3
```

You can display any number of items in the `print()` function using the following syntax:

```
print(item1, item2, ..., itemk)
```

Example 05: Displays the text "The answer is" and the value of the variable answer.

```
x = 5
y = 2
answer = x * y

print("The answer is:", answer)
```

Output

```
The answer is: 10
```

2.4 Input and Output

In Python, we use the `input()` function to request an input from the user. It prompts for the use input, coverts the input into a string, then returns that.

Example 06: Using the `input()` function

```
name = input("Enter your name: ")

print("Hello ", name)
```

Output

```
Enter your name: John
Hello John
```

The `input()` function will return whatever is entered as a string. Numeric values can be used in calculations but strings cannot. You can use the function `eval()` to evaluate and convert it to a numeric value. For example,

There are three functions that can be used to converting a string into a number:

- `int()` for whole numbers.
- `float()` for floating-point numbers.
- `eval()` to evaluate an expression.

`int("3")` returns `3`
`int("2.4")` will cause an error

`float("3")` returns `3.0`
`float("2.4")` returns `2.4`

`eval("3")` returns `3`
`eval("2.4")` returns `2.4`
`eval("3 + 4")` returns `7`

Example 06: Using **`eval()`** function.

```
number1 = eval(input("Enter a number1: "))
number2 = eval(input("Enter a number1: "))

print("number1 + number2 =", number1 + number2)
```

Output

```
Enter number1: 2
Enter number2: 3.8
number1 + number2 = 5.8
```

2.5 Numeric Operators

Python provides arithmetic operators as shown in a table below.

TABLE 2.1 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`

2.6 Augmented Assignment Operators

The arithmetic operators can be combined with the assignment operator (**=**) to form augmented assignment operators.

TABLE 2.2 Augmented Assignment Operators

Operator	*Name*	*Example*	*Equivalent*
`+=`	Addition assignment	`i += 8`	`i = i + 8`
`-=`	Subtraction assignment	`i -= 8`	`i = i - 8`
`*=`	Multiplication assignment	`i *= 8`	`i = i * 8`
`/=`	Float division assignment	`i /= 8`	`i = i / 8`
`//=`	Integer division assignment	`i //= 8`	`i = i // 8`
`%=`	Remainder assignment	`i %= 8`	`i = i % 8`
`**=`	Exponent assignment	`i **= 8`	`i = i ** 8`

Note: There are no spaces in the augmented assignment operators. For example, + = should be +=.

2.7 Importing Modules

Python, as well as other programming languages, provides a library of functions. You have already used the functions `print()`, `input()`, and `eval()`. These are built-in functions and they are always available in the Python interpreter. You do not have to **import** any library module to use these functions.

As for library functions, for example, Mathematical functions, which are NOT already available in Python interpreter, you will need to `import` the `math` module at the start of your program in order to use these functions.

An example below is a program that tests some math functions. Because the program uses the math functions defined in the math module, the math module is imported in line 1.

Example 07: Using math functions

```
import math # import math module to use the math functions

# Test algebraic functions
print("exp(1.0) =", math.exp(1))
print("log(2.78) =", math.log(math.e))
print("log10(10) =", math.log10(10))
print("sqrt(4.0) =", math.sqrt(4.0))
```

Output:

```
exp(1.0) = 2.718281828459045
log(2.78) = 1.0
log10(10) = 1.0
sqrt(4.0) = 2.0
```

Use `from` ... `import` ... when you want to save yourself from typing the module name over and over again.

Example 08: Using math functions

```
from math import * # import everything in the math module

# Test trigonometric functions
print("sin(PI / 2) =", sin(pi / 2))
print("cos(PI / 2) =", cos(pi / 2))
print("tan(PI / 2) =", tan(pi / 2))
print("degrees(1.57) =", degrees(1.57))
print("radians(90) =", radians(90))
```

Output:

```
sin(PI / 2) = 1.0
cos(PI / 2) = 6.123233995736766e-17
tan(PI / 2) = 1.633123935319537e+16
degrees(1.57) = 89.95437383553924
radians(90) = 1.5707963267948966
```

Practice

Write the following programs:

1. A runner runs `14` kilometers in `45` minutes and `30` seconds. Write a program that displays the average speed in miles per hour. (Note that `1` mile is `1.6` kilometers.). Round the result to two decimal places.
2. There are `2.204` pounds in a kilogram. Write a program to ask the user to enter a weight in kilograms and convert it to pounds. Round the result to two decimal places.

3. Boolean Expressions

A Boolean expression is an expression that evaluates to a Boolean value `True` or `False`. Python provides six comparison operators (also known as relational operators), shown in Table 4.1

TABLE 4.1 Comparison Operators

Python Operator	*Name*	*Example* `(radius is 5)`	*Result*
`<`	less than	`radius < 0`	`False`
`<=`	less than or equal to	`radius <= 0`	`False`
`>`	greater than	`radius > 0`	`True`
`>=`	greater than or equal to	`radius >= 0`	`True`
`==`	equal to	`radius == 0`	`False`
`!=`	not equal to	`radius != 0`	`True`

4. Logical Operators

Sometimes, a combination of several conditions determines whether a statement is executed. You can use logical operators `not`, `and`, and `or` can be used to create a composite condition.

Logical operators, also known as *Boolean operators*, operate on Boolean values to create a new Boolean value.

- `not` operator negates `True` to `False` and `False` to `True`.
- `and` of two Boolean operands is true if and only if both operands are true.
- `or` of two Boolean operands is true if at least one of the operands is true.

Practice

Assuming that **x** is **1**, show the result of the following Boolean expressions.

1). `True and (3 > 4)`
2). `not (x > 0) and (x > 0)`
3). `(x > 0) or (x < 0)`
4). `(x != 0) or (x == 0)`
5). `(x >= 0) or (x < 0)`
6). `(x != 1) == not (x == 1)`
7). `2 * 2 - 3 > 2 and 4 - 2 > 5`
8). `2 * 2 - 3 > 2 or 4 - 2 > 5`

5. Chained Comparisons

Comparison operators can be chained together to arbitrary length. For example, the following expressions are equivalent:

`x < y <= z` is equivalent to `x < y and y <= z`

Practice

Assuming x = 4 and y = 5, show the result of the following Boolean expressions:

1). `x >= y >= 0`
2). `x <= y >= 0`
3). `x != y == 5`
4). `5 >= 3 != 10 > 4 + 3 < 20 <= 5 > 3`

6. Selection Statements

Selection statements lets program decide which statements to execute based on a condition. Python has several types of selection statements such as:

- `if` statements
- `if-else` statements
- `if-elif-else` statements
- Conditional Expression

6.1 `if` Statements

```
if condition:
    # statements will execute if the condition is True
```

Example 09: Using `if` statements.

```
number = eval(input("Enter an integer: "))

if number >= 0:
    print("The number is positive")
```

Output:

```
Enter an integer: 4
The number is positive
```

The body of the `if` statement must be indented at least one space to the right of the **if** keyword, and each statement must be **indented using the same number of spaces**. Most common errors in selection statements are caused by incorrect indentation.

6.2 if-else Statements

```
if condition:
    # statements will execute if the condition is True

else:
    # statements will execute if the condition is False
```

Example 10: Using if-else statement

```
number = eval(input("Enter an integer: "))

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

Output:

```
Enter an integer: 4
4 is even.
```

6.3 if-elif-else Statements

```
if condition:
    # statements will execute if the condition is True
elif condition:
    # statements will execute if the condition is True
else:
    # statements will execute if all conditions above are False
```

Example 11: Using if-elif-else statement.

```
score = eval(input("Enter a score: "))

if score >= 85:
    grade = "A"
elif score >= 75:
    grade = "B"
elif score >= 65:
    grade = "C"
elif score >= 50:
    grade = "D"
else:
    grade = "F"
print("Grade: " , grade)
```

Output:

```
Enter a score: 72
Grade C
```

Practice

Write the following programs:

1. Ask a user to enter a number that is under `20`. If they enter a number that is `20` or more, display the message `"Too high"`, otherwise display `"Thank you"`.

2. Ask the user to enter a number. If it is under 10, display the message `"Too low"`, if their number is between `10` and `20`, display `"Correct"`, otherwise display `"Too high"`.

6.4 Conditional Expressions

Conditional expression, aka. Ternary operator, is used to write an `if-else` statement in one line.

Syntax:

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

For example, you might want to assign a value to a variable that is restricted by certain conditions:

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

Alternatively, you can use a conditional expression to achieve the same result:

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

Practice

1. Rewrite the following **if** statements using a conditional expression:

```
if ages >= 16:
    ticketPrice = 20
else:
    ticketPrice = 10
```

2. Rewrite the following program using a conditional expression:

```
age = eval(input("Enter an age: "))

if age >= 18:
    remark = "Adult"
else:
    remark = "Kid"

print(remark)
```

Example 12: The following statement displays the message **number is even** if **number** is even, and otherwise, displays **number is odd**.

```
number = eval(input("Enter an integer: "))

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

Output:

```
Enter an integer: 11
number is odd
```

Output:

```
Enter an integer: 10
number is even
```

7. `pass` Statement

When the user does not know what code to write, so user simply places `pass` at that line. Sometimes, `pass` is used when the user does not want any code to execute. So, the user can simply place `pass` where empty code is not allowed, like in loops, function definitions, class definitions, or in if statements. So using `pass` statement user avoids this error.

Example 13: Using `pass` statement

```
number = 5

if number % 2 == 0:
    pass
else:
    print(number, "is not divisible by 2")
```

Output:

```
5 is not divisible by 2
```

8. Loops

In python, there are two types of loops: `while` loop and `for` loop.

8.1 The `while` Loop

A while loop executes statements repeatedly as long as a condition remains true.

The syntax for the while loop is:

```
while condition:
    # Loop body
```

Example 14: A program to calculate: 1 + 2 + 3 + … + 10 using `while` loop.

```
total = 0
i = 1

while i <= 10:
    total += i
    i += 1

print("Total =", total)
```

Output

```
Total = 55
```

Practice

Write a program to calculate the following series:

$$\frac{1}{3} + \frac{3}{5} + \frac{5}{7} + \frac{7}{9} + \frac{9}{11} + \frac{11}{13} + \ldots + \frac{95}{97} + \frac{97}{99}$$

8.2 The `for` Loop

A `while` **loop** allows code to be repeated an unknown number of times as long as a condition is being met, while a `for` loop allows code to repeated a set number of times. `for` loop is sometimes known as a **counting loop** because you know the number of times the loop will run before it starts. In general, the syntax of a `for` loop is:

```
for var in sequence:
    # Loop body
```

A **sequence** holds multiple items of data, stored one after the other. The variable **var** takes on each item in the sequence, and the statements in the loop body are executed once for each item.

Example 15:

```
for v in range(4, 8):
    print(v, end = "\t")
```

Output:

```
4       5       6       7
```

By default, python's `print()` function ends with a newline. `end = "\t"` causes a **Tab** to be printed after each number, rather than the default newline character \n.

There are three versions of the `range()` function:

- `range` **(a, b)** returns the sequence of integers **a**, **a + 1**, ..., **b - 2**, and **b - 1**.
- `range` **(a)** is the same as `range`**(0, a).**
- `range` **(a, b, k)**. The first number in the sequence is **a**. Each successive number in the sequence will increase by **k** which is a *step value*. **b** is the limit. The last number in the sequence is less than **b**.

Example 16:

```
for v in range(3, 9, 2):
    print(v, end = "\t")
```

Output:

```
3       5       7
```

8.3 Loop Control Statements

The `break` and `continue` keywords provide additional controls to a loop.

8.3.1 The `break` Keyword

You can use the keyword **break** in a loop to immediately terminate a loop.

Example 17: Using `break`.

```
count = 0

while count < 5:
    if count == 3:
        break

    print(count)
    count += 1
```

Output:

```
0
1
2
```

Without `if count == 3: break`, this program would print out the numbers from `0` to `4`. But with `if count == 3: break`, the loop terminates when `count` becomes `3`.

8.3.2 The `continue` Keyword

You can use the `continue` keyword to end the current iteration and make the program control goes to the end of the loop body. In other words, `continue` breaks out of an iteration, while the `break` keyword breaks out of a loop.

Example 18: Using `continue`.

```
count = 0

while count < 5:
    if count == 3:
        count += 1
        continue

    print(count)
    count += 1
```

Output:

```
0
1
2
4
```

The `continue` statement is executed when `count` becomes `3`. The `continue` statement ends the current iteration so that the rest of the statement in the loop body is not executed; therefore, `3` is not printed out.

Practice

Write a program that asks the user to enter their name and a number. If the number is less than `10`, then display their name that number of times; otherwise display the message "Too high" three times.

8.4 The `for-else` Loop

In other languages, the `else` statement is only used with an `if` statement. But Python allows us to use the `else` statement with a `for` loop statement as well.

If the `else` statement is used with a `for` loop, the `else` block will be executed when the loop is finished.

Example 19: Using `else` with a `for` loop.

```
for x in range(5):
    print(x, end = " ")
else:
    print("Finally finished!")
```

Output

```
0 1 2 3 4 Finally finished!
```

The `else` block will NOT be executed if the loop is terminated by a `break` statement. Let's look at an example below:

Example 20:

```
for x in range(5):
    if x == 3:
        break
    print(x, end = " ")
else:
    print("Finally finished!")
```

Output

```
0 1 2
```

In a nested loop, a **break** statement only exits the loop it is placed in. To create a multi-level **break** statements (that exits both inner loop and outer loop), you can do as below:

Example 21: Break multiple loops at once

```
for i in range(4):
    for j in range(3):
        print(i, "and", j)
        if i == 2:
            break  # Break the inner loop...
    else:
        continue  # Continue if the inner loop was not broken
    break  # Inner loop was broken, break the outer
```

Output

```
0 and 0
0 and 1
0 and 2
1 and 0
1 and 1
1 and 2
2 and 0
```

9. Strings

A **string** is a sequence of characters. String values can be enclosed in matching single quotes (') or double quotes (").

Example 22: Concatenate strings.

```
string = "Welcome " + "to " + "Python"
print (string)
```

Output:

```
Welcome to Python
```

9.1 The `str`() Function

Convert an integer into string:

`str(3)` returns `"3"`

Convert a float into string:

`str(3.4)` returns `"3.4"`

Example 23: Ask the user to answer an addition.

```
num1 = 2
num2 = 5

answer = eval(input("What is " + str(num1) + " + " + str(num2) + "? "))

print(answer == num1 + num2)
```

Output

```
What is 2 + 5? 7
True
```

9.2 Multiplying Strings

Multiplication operator is used with strings in Python for the purpose of repetition.

Multiplication operator when used with string in Python creates new strings by concatenating multiple copies of the same string.

Example 21: Multiply a string by five times.

```
print("Hello " * 5)
```

Output

```
Hello Hello Hello Hello Hello
```

Note that the value that multiply with the string must be an integer, or else, you will get an error.

9.3 String 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.

The syntax to invoke a method for an object is `object.method()`

TABLE 3.3 Simple String Methods:

Method	Description	Example	Result
`lower()`	Return a new string in lower case.	`"Hello".lower()`	hello
`upper()`	Return a new string in upper case.	`"Hello".upper()`	HELLO
`capitalize()`	Return a new string in which the first letter of the string is capitalized and everything else is in lower case.	`"hello".capitalize()`	Hello
`title()`	Return a new string in which the first letter of each word in the string is capitalized and everything else is in lower case.	`"hello python".title()`	Hello Python
`strip()`	Removes specified characters at the start and end of a string.	`"  Hello Python  ".strip(" ")`	Hello Python
`count()`	Returns the number of occurrences of the substring in the given string.	`"Hello Python".count("o")` `"Hello Python".count("on")`	2 1
`replace()`	Returns a copy of the string in which the occurrences of the old characters in the string have been replaced with new ones.	`"Hello C".replace("C", "Python")`	"Hello Python"
`split()`	splits a string into a list by specify a separator, default separator is any whitespace.	`str1, str2 = "Hello, Welcome!".split(", ")` `print(str2)`	Welcome!

More string methods here: https://www.programiz.com/python-programming/methods/string

Example 24: Using string methods.

```
s = "welCome tO pyThoN"
s1 = s.lower() # Call the lower method
print(s1)

s2 = s.upper() # Call the upper method
print(s2)

s3 = s.capitalize()  # Call the capitalize method
print(s3)

s4 = s.title()  # Call the title method
print(s4)
```

Output

```
welcome to python
WELCOME TO PYTHON
Welcome to python
Welcome To Python
```

9.4 Accessing String Characters

You can access the character of a string using index. The first character has index `0`. Negative indexing means beginning from the end, `-1` refers to the last character, `-2` refers to the second last character, etc.

Example 25: Access a string's character by referring to the index number.

```
string = "Hello Python"

print(string[0])
print(string[1])
print(string[-1])
print(string[-2])
```

Output

```
H
e
n
o
```

H	e	l	l	o		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

9.5 Iterating Through a String

As strings are sequences, you can loop through them directly.

Example 26: Loop through a string.

```
for i in "Hello":
    print(i, end = "\t")
```

Output

```
H	e	l	l	o
```

You also can loop through a string by index.

Example 27: Loop through a string by index.

```
string = "Hello"

for i in range(len(string)):
    print(string[i], end = "\t")
```

Output

```
H       e       l       l       o
```

Practice

Write a program that asks the user to enter their last name in lower case, and then ask them to enter their first name in lower case. Change the case of the last name to upper case, and capitalize the first name. Then, join them together with a space between and display the whole name and the length of whole name.

Exercises

1. Write a program that calculates the energy needed to heat water from an initial temperature to a final temperature. Your program should ask the user to enter the amount of water in kilograms and the initial and final temperatures of the water. The formula to compute the energy is:

   Q = M * (final_temperature – initial_temperature) * `4184`

   where **M** is the weight of water in kilograms, temperatures are in degrees Celsius, and energy **Q** is measured in joules.

2. Displays the following message:

   ```
   1) Square
   2) Triangle

   Enter a number:
   ```

   If the user enters 1, then it should ask them for the length of one of its sides and display the area. If they select 2, it should ask for the base and height of the triangle and display the area. If they type in anything else, it should give them a suitable error message.

3. Write a program to display Body mass index (BMI). BMI is a measure of health based on weight. It can be calculated by taking your weight in kilograms and dividing it by the square of your height in meters. The interpretation of BMI for people 16 years and older is as follows:

BMI	**Interpretation**
Below 18.5	Underweight
18.5 – 24.9	Normal
25.0 – 29.9	Overweight
30.0 or Above	Obese

4. Calculate: 1 - 2 + 3 - 4 + … + 29 - 30
5. Create a variable called computer_num and set the value to 50. Ask the user to enter a number. While their guess is not the same as the computer_num value, tell them if their guess is too low or too high and ask them to have another guess. If they enter the same value as computer_num, display the message "Well done, you took [count] attempts".
6. (*Find numbers divisible by 5 or 6, but not both*) Write a program that displays ten numbers per line, all the numbers from 100 to 200 that are divisible by 5 or 6, but not both. The numbers are separated by exactly one space.
7. Ask the user to enter their favorite color. If they enter "red", "RED" or "Red" display the message "I like red too", otherwise display the message "I don't like [color], I prefer red".
8. How often does each type of these 20 amino acids: A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W and Y occurs in the below protein sequence:

   GIVEQCCTSICSLYQLENYCNFVNQHLCGSHLVEALYLVCGERGFFYTPKTNQHERGFFYTP
   KSICSLYQLVCGEVEQCCTTSICSLYLCGSHRGFFYTLVECGEALYLHERGICSLYQLENYCN
   FVNQHLCGSHLVEALYLVCGERGFFYTPKTNQHERGFFYTPKSICSLYQLVCGEVEQCCTTS
   ICSLYLCGSQCCTTSICSLYLCGSHRGFFYTLVECGEALYLHERGICSLYQLENYCNFVNQHL