Introduction to

PYTHON PROGRAMMING

Dr. Thyagaraju G S

Iterative International Publishers

IIP

Introduction to Python Programming

First Edition

Author
Dr. Thyagaraju G. S

Title of the Book: Introduction to Python Programming

Edition First: 2023

ISBN: 978-93-5747-477-1

MRP Rs. 590/-

Publisher, Printed at & Distribution by:
Selfypage Developers Pvt Ltd.,
Pushpagiri Complex,
Beside SBI Housing Board,
K.M. Road Chikkamagaluru, Karnataka.
Tel.: +91-8861518868
E-mail:publish@iiponline.org
IMPRINT: I I P Iterative International Publishers

For Sales Enquiries
Contact: 91- 8861511583
E-mail: sales@iiponline.org

Dedicated to Context

Preface

Welcome to the world of Python programming! This book, titled "Introduction to Python Programming," is designed to be your comprehensive guide to learning the fundamentals of Python, one of the most popular and versatile programming languages in the world.

Python is known for its simplicity, readability, and ease of use, making it an excellent choice for beginners and experienced programmers alike. Whether you are a complete novice or have some programming experience in other languages, this book will take you on a journey to master the basics of Python and explore its powerful features.

Chapter 1.1: **Python Basics**

In this chapter, you'll start to learn the absolute basics. You'll learn how to enter expressions into the interactive shell, work with different data types such as integers, floating-point numbers, and strings. We'll also explore string concatenation, storing values in variables, and writing your first Python program. By dissecting your program, you'll gain a deeper understanding of how Python code is executed.

Chapter 1.2: **Flow Control**

Flow control is essential for making decisions in your code. Here, you'll delve into Boolean values, comparison operators, and Boolean operators. These concepts is mixed to create conditional statements, loops, and other flow control structures. Additionally, you'll discover how to import modules and end a program early when necessary.

Chapter 1.3: **Functions**

Functions allow you to organize and reuse code effectively. You'll learn how to define functions with parameters and return values. We'll also explore the special value "None," keyword arguments, and working with local and global scope. Exception handling will be introduced to handle errors gracefully. Finally, we'll apply these concepts to create two exciting projects: "Guess the Number" and "Password Locker."

Chapter 2: **Lists, Strings, Tuples and Dictionaries**

Lists and dictionaries are fundamental data structures in Python. You'll explore lists and their various methods, including working with strings and tuples, as well as references. Next, we'll delve into dictionaries and how they allow us to structure data efficiently. You'll apply these concepts to build a fun project called the "Magic 8 Ball."

Chapter 3: **Manipulating Strings and Reading/Writing Files**

Understanding strings is vital, as they play a significant role in many programs. You'll explore useful string methods and work with files, including reading and writing data to files. You'll gain valuable insights through projects like "Password Locker" and "Adding Bullets to Wiki Markup."

Chapter 4: **Organizing Files and Debugging**

Learn how to organize files with the "shutil" module and manipulate directories with the "os.path" module. We'll explore debugging techniques using exceptions, assertions, and the IDLE debugger.

Chapter 5.1: **Classes and Objects**

Object-oriented programming is a powerful paradigm in Python. In this chapter you'll discover how to create your own classes and work with attributes and instances. You'll explore the concept of objects as mutable entities and learn about copying objects.

Chapter 5.2: Classes and Methods

In this chapter, you'll explore classes and methods. Understand the concept of pure functions and modifiers and learn about prototyping versus planning.

Chapter 5.2: Classes and Functions

In this chapter, you'll explore object-oriented features such as operator overloading, type-based dispatch, and polymorphism. The concept of interface and implementation will be discussed to design maintainable and flexible code.
I hope this book will serve as your comprehensive resource for learning Python programming. Each chapter is designed to build upon the knowledge from the previous ones, ensuring a smooth and rewarding learning experience. Whether you aspire to develop web applications, data analysis tools, or explore machine learning, Python will be your reliable companion.

To access supplementary learning materials related to "**GUI Programming using Python**" and to procure Python source code and a comprehensive collection of **e-materials** associated with the book, I encourage you to visit **my website**: **https://tocxten.com/ .**

Happy Learning!
Dr. Thyagaraju G. S

Acknowledgements

I express my gratitude to my daughter, Palguni GT, who is currently pursuing B.E. in Computer Science and Business Systems at Malnad College of Engineering (MCE), Hassan. Her invaluable assistance in composing the book is deeply appreciated. Her contributions included coding and testing the necessary Python programs and compiling the question bank.

……..**Dr. Thyagaraju G. S**

Contents

Chapter
1

Python Basics, Flow Control and Functions

1.1 Python Basics

Topics Covered: Entering Expressions into the Interactive Shell, The Integer, Floating-Point, and String Data Types, String Concatenation and Replication, Storing Values in Variables, Your First Program, Dissecting Your Program.

1.1.1 Introduction

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985 - 1990. Python is named after a TV Show called ‘Monty Python’s Flying Circus’ and not after Python-the snake. Some of the Features that Makes Python more popular are:

- Python is Simple and Easy to learn and code.
- Python is Free and Open Source. It is freely available at the **https://www.python.org/**. Python source code is also available to the public, one can download it, use it, and share it.
- Python is High Level Language and supports both Procedure oriented and Object-Oriented Language concepts along with dynamic memory management.
- Python is portable. Python code can be run on any platforms like Linux, Unix, Mac and Windows.
- Python is extensible and integrated. Python code can be extended and integrated with other languages like C, C++, Java, etc.
- Python is an interpreted language. Python code is executed line by line at a time and there is no need to compile, which makes debugging easier.
- Python has rich set of libraries for data analytics, machine learning, artificial intelligence, deep learning, mathematical computation, web app development, mobile app development, testing, etc.

- Python is a dynamically typed language. Here the data type for variable is decided at run time. As a result, there is no need to specify the type of variable.

1.1.2 Entering Expressions into the interactive Shell

Entering Python expressions into the interactive shell on a Windows system is quite straightforward. The process involves opening the Command Prompt (CMD) and invoking the Python interpreter in interactive mode. Here's a step-by-step explanation with examples:

Open Command Prompt: Press the Windows key to open the start menu, then type "cmd" and press Enter. This will open the Command Prompt.

Start the Interactive Python Shell: In the Command Prompt, type python and press Enter. This will launch the Python interpreter in interactive mode.

Input Python Expressions: Once you're in the interactive shell (the >>> prompt), you can start entering Python expressions.

For example, you can do basic arithmetic calculations

```
>>> 5 + 3
8
>>> 10 * 2
20
>>> 15 / 3
5.0
```

You can also use variables

```
>>> x = 10
>>> y = 5
>>> x + y
15
```

And perform more complex operations:

```
>>> import math
>>> radius = 5
>>> area = math.pi * radius**2
>>> area
78.53981633974483
```

Multi-line Statements: Similar to the examples provided in the previous response, you can enter multi-line statements or loops in the interactive shell. Just remember to use proper indentation for code blocks.

Exiting the Interactive Shell: To exit the interactive shell and return to the Command Prompt, you can type exit(), quit(), or press Ctrl + Z followed by Enter. **Here's how it looks in practice:**

```
C:\Users\YourUsername>python
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct  5 2020, 15:34:40) [MSC v.1927 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 5 + 3
8
>>> x = 10
>>> y = 5
>>> x + y
15
>>> import math
>>> radius = 5
>>> area = math.pi * radius**2
>>> area
78.53981633974483
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
>>> exit()
```

Note: *All the remaining programs used in the book are executed and tested in Jupyter Notebook.*

1.1.3 Expressions

In Python, expressions are combinations of values, variables, operators, and function calls that can be evaluated to produce a result. They represent computations and return a value when executed. Here are some examples of expressions in Python:

Examples: 17, x, x+17, 1+2*2, X**2, x**2 + y**2
Entering expressions into the Python interactive shell/Jupyter notebook cell allows you to evaluate and execute code in real-time. Here are some more examples of expressions:

- **Arithmetic Operations:** You can perform basic arithmetic operations, such as addition, subtraction, multiplication, and division, directly in the shell. For example:
  2 + 3 **#Output:** 5

  4 * 5 **#Output:** 20

  10 / 3 **#Output:** 3.3333333333333335

- **Variable Assignment:** You can assign values to variables and use those variables in expressions. For example:
  x = 5
  y = 2
  x + y **#Output:** 7

  x * y **#Output:** 10

- **Function Calls**: You can call built-in functions or user-defined functions within the shell. For example:
  abs(-10) **#Output:** 10

  len("Hello, World!") **#Output:** 13

  ```
  def add(a, b):
      return a + b
  ```
  add(3, 4) **#Output:** 7

- **Boolean Expressions:** You can use Boolean operators like and, or, and not to evaluate logical expressions. For example:

  True and False #**Output:** False

  True or False **#Output:** True

  not True **# Output:** False

- **Conditional Statement:** You can use conditional statements like if, elif, and else to perform different actions based on conditions. For example:

```
x = 10
if  x > 0:
    print("Positive")
elif  x < 0:
     print("Negative")
else:
    print("Zero")
```

  #output: Positive

1.1.4 Value

A value is a letter or a number. In Python, a value is a fundamental piece of data that can be assigned to variables, used in expressions, and manipulated by operations. Values can be of different types, such as numbers, strings, Booleans, lists, tuples, dictionaries, and more. Each type of value has its own characteristics and behaviors.

Examples

```
x = 10                          # integer
y = 3.14                        # floating-point number
z = 2 + 3j                      # complex number
name = "John"                   # string
message = 'Hello, World!'       # string
is_true = True                  # Boolean Value
is_false = False
numbers = [1, 2, 3, 4, 5]
fruits = ["apple", "banana", "orange"]
coordinates = (10, 20)
person = ("John", 30, "USA")
student = {"name": "John", "age": 20, "grade": "A"}
```

1.1.5 type() function

Types are the data types to which the values belong. In Python, the **type()** function is used to determine the type of a given object or value. It returns the data type of the object as a result. The **type()** function is particularly useful when you need to programmatically determine the type of a variable or check if a value belongs to a specific data type.

This line assigns a new list ['bat', 'rat', 'cow'] to the variable l1. Since a new list object is assigned to l1, its identifier changes, and the output of id(l1) will be a new memory address (e.g., 2626115265152).

In summary, these code snippets illustrate the concepts of unique object identifiers (memory addresses), string immutability, and list mutability in Python.

3. **Pass by Reference:** In Python, all arguments (parameters) are passed by assignment. This concept is often referred to as "pass by assignment" or "pass by object reference." It's important to note that Python doesn't use the terms "pass by value" or "pass by reference" in the same way as some other programming languages do. Instead, it operates on the idea of binding names (variables) to objects.

   Here's how "**pass by reference**" works in Python, along with an example: When you pass an argument to a function, what you're actually passing is a reference to an object. This reference is essentially a pointer that allows the function to access the original object. If the object is mutable, like a list, changes made to the object within the function will be reflected outside the function as well. However, if the object is immutable, like a string or integer, any changes made to it within the function will not affect the original object.

   Let's illustrate this with an example:

   ```
   def modify_list(lst):
      lst.append(4)
      lst[0] = 100

   my_list = [1, 2, 3]
   modify_list(my_list)
   print(my_list)  # Output: [100, 2, 3, 4]
   ```

   In this example, **my_list** is a list that gets passed to the **modify_list** function. Within the function, the list's elements are modified. Since lists are mutable, the changes made inside the function persist outside as well. As a result, when we print my_list after calling the function, it reflects the modifications made within the function.

   It's important to understand that even though we say "pass by reference" in Python, it's not the same as true "pass by reference" as seen in some other languages. In Python, the reference itself is passed by value, which means you can't change the reference to point to a different object within the function and have it affect the original reference outside the function.

In summary, while Python uses the idea of passing references to objects, it doesn't fit precisely into the "pass by value" or "pass by reference" categorizations of other languages. Instead, it's more accurately described as **"pass by assignment" or "pass by object reference."**

4. **Slicing of nested Lists:** Consider a nested list, **my_list = [10, [25,30],[["Hello",True,False,[-2,-3,4]]]]**
   Some of the possible slices and their values are as listed below:

	Slice	Value
1.	my_list[:]	[10, [25, 30], [['Hello', True, False, [-2, -3, 4]]]]
2.	my_list[0:]	[10, [25, 30], [['Hello', True, False, [-2, -3, 4]]]]
3.	my_list[1:]	[[25, 30], [['Hello', True, False, [-2, -3, 4]]]]
4.	my_list[2:]	[[['Hello', True, False, [-2, -3, 4]]]]
5.	my_list[:1]	[10]
6.	my_list[:2]	[10, [25, 30]]
7.	my_list[:3]	[10, [25, 30], [['Hello', True, False, [-2, -3, 4]]]]
8.	my_list[0:1]	[10]
9.	my_list[0:2]	[10, [25, 30]]
10.	my_list[0:3]	[10, [25, 30], [['Hello', True, False, [-2, -3, 4]]]]
11.	my_list[1:2]	[[25, 30]]
12.	my_list[1:3]	[[25, 30], [['Hello', True, False, [-2, -3, 4]]]]
13.	my_list[2:3]	[[['Hello', True, False, [-2, -3, 4]]]]
14.	my_list[0][0:]	10
15.	my_list[0][:1]	10
16.	my_list[0][0:1]	10
17.	my_list[1][0:]	[25, 30]
18.	my_list[1][:1]	[25, 30]
19.	my_list[1][0:1]	[25, 30]
20.	my_list[2][0:]	[['Hello', True, False, [-2, -3, 4]]]
21.	my_list[2][:1]	[['Hello', True, False, [-2, -3, 4]]]
22.	my_list[2][0:1]	[['Hello', True, False, [-2, -3, 4]]]
23.	my_list[2][0][0:]	['Hello', True, False, [-2, -3, 4]]
24.	my_list[2][0][:1]	['Hello']
25.	my_list[2][0][0:1]	['Hello']
26.	my_list[2][0][1:]	[True, False, [-2, -3, 4]]
27.	my_list[2][0][1:3]	[True, False]
28.	my_list[2][0][2:]	[False, [-2, -3, 4]]
29.	my_list[2][0][2:4]	[False, [-2, -3, 4]]

30.	my_list[2][0][3:]	[[-2, -3, 4]]
31.	my_list[2][0][3:4]	[[-2, -3, 4]]
32.	my_list[2][0][3][0:]	-2
33.	my_list[2][0][3][1:]	-3
34.	my_list[2][0][3][1:2]	-3
35.	my_list[2][0][3][2:]	4

5. Python Keywords

Sl.No	Keyword	Meaning
1	False	Boolean value representing false.
2	None	Represents the absence of a value or null.
3	True	Boolean value representing true.
4	and	Logical operator representing 'and'.
5	as	Used for creating aliases while importing modules.
6	assert	Used for debugging purposes to check conditions.
7	async	Declares a function to be asynchronous.
8	await	Used within an asynchronous function to pause.
9	break	Breaks out of the current loop.
10	class	Defines a class.
11	continue	Skips the rest of the current loop iteration.
12	def	Defines a function.
13	del	Deletes a reference or element.
14	elif	Used in conditional statements, like 'else if'.
15	else	Used in conditional statements for fallback.
16	except	Catches exceptions in try/except blocks.
17	finally	Executes code after a try/except block.
18	for	Used for looping over iterable objects.
19	from	Used to import specific attributes or modules.
20	global	Declares a variable to have global scope.
21	if	Conditional statement for decision-making.
22	import	Imports modules or attributes into the script.
23	in	Used to check membership in an iterable.
24	is	Checks if two variables refer to the same object.
25	lambda	Creates an anonymous (inline) function.
26	nonlocal	Declares a variable to be non-local in scope.
27	not	Logical operator representing 'not'.
28	or	Logical operator representing 'or'.
29	pass	Placeholder statement with no action.
30	raise	Raises an exception in code.
31	return	Exits a function and returns a value.
32	try	Starts a block of code to be tested for exceptions.

33	while	Creates a loop that continues while a condition is true.
34	with	Creates a context manager for resource management.
35	yield	Used in generator functions to yield a value.

6. **Shallow Copy and Deep Copy:** In Python, when you work with objects like lists, you often need to make copies of them for various purposes. There are two main ways to create copies of lists: shallow copy and deep copy. Let's explore these concepts using nested lists as examples and the id() function to demonstrate the differences.

   a. **Shallow Copy:** A shallow copy creates a new object that is a copy of the original list. However, if the list contains nested objects (like other lists), it only copies the references to those objects, not the objects themselves. In other words, the copy references the same nested objects as the original list.

   Here's an example:

   ```
   import copy
   original_list = [[1, 2], [3, 4]]
   shallow_copied_list = copy.copy(original_list)

   # Check if the outer lists are the same
   print(original_list == shallow_copied_list)  # True

   # Check if the outer lists have different memory addresses
   print(id(original_list) == id(shallow_copied_list))  # False

   # Check if the nested lists are the same
   print(original_list[0] == shallow_copied_list[0])  # True
   print(original_list[0] == shallow_copied_list[0])  # True

   # Check if the nested lists have different memory addresses
   print(id(original_list[0]) == id(shallow_copied_list[0]))  # True
   print(id(original_list[1]) == id(shallow_copied_list[1]))  # True
   ```

   As you can see, the shallow copy creates a new list with a different memory address for the outer list, but it references the same nested lists.

   b. **Deep Copy:** A deep copy, on the other hand, creates a new object that is a complete and independent copy of the original list, including all nested

objects. This ensures that changes made to the deep copy do not affect the original list or its nested objects.
Here's an example:

```
import copy

original_list = [[1, 2], [3, 4]]
deep_copied_list = copy.deepcopy(original_list)

# Check if the outer lists are the same
print(original_list == deep_copied_list)  # True

# Check if the outer lists have different memory addresses
print(id(original_list) == id(deep_copied_list))  # False

# Check if the nested lists are the same
print(original_list[0] == deep_copied_list[0])  # True
print(original_list[1] == deep_copied_list[1])  # True
# Check if the nested lists have different memory addresses
print(id(original_list[0]) == id(deep_copied_list[0]))  # False
print(id(original_list[1]) == id(deep_copied_list[1]))  # False
```

7. Sources of information

[1] Al Sweigart, “Automate the Boring Stuff with Python”,1stEdition, No Starch Press, 2015. (Available under CC-BY-NC-SA license at https://automatetheboringstuff.com/)
[2] https://www.learnbyexample.org/python-lambda-function/
[3] Allen B. Downey, “Think Python: How to Think Like a Computer Scientist”, 2nd Edition, Green Tea Press, 2015. (Available under CC-BY-NC license at http://greenteapress.com/thinkpython2/thinkpython2.pdf
[4] https://www.learnbyexample.org/python/
[5] https://www.learnpython.org/
[6] https://pythontutor.com/visualize.html#mode=edit
[7] https://www.learnbyexample.org/python/
[8] https://www.learnpython.org/
[9] https://pythontutor.com/visualize.html#mode=edit
[10] https://github.com/sushantkhara/Data-Structures-And-Algorithms-with-Python/raw/main/Python%203%20_%20400%20exercises%20and%20solutions%20for%20beginners.pdf
[11] https://www.python.org/
[12] https://www.anaconda.com/
[13] https://chat.openai.com/

[14] Datatypes: https://www.youtube.com/watch?v=gCCVsvgR2KU
[15] Operators: https://www.youtube.com/watch?v=v5MR5JnKcZI
[16] Flow Control: https://www.youtube.com/watch?v=PqFKRqpHrjw
[17] For loop: https://www.youtube.com/watch?v=0ZvaDa8eT5s
[18] While loop: https://www.youtube.com/watch?v=HZARImviDxg
[19] Exceptions: https://www.youtube.com/watch?v=6SPDvPK38tw
[20] Functions: https://www.youtube.com/watch?v=BVfCWuca9nw
[21] Arguments: https://www.youtube.com/watch?v=ijXMGpoMkhQ
[22] Return value: https://www.youtube.com/watch?v=nuNXiEDnM44
[23] Strings: https://www.youtube.com/watch?v=lSItwlnF0eU
[24] String functions: https://www.youtube.com/watch?v=9a3CxJyTq00
[25] Lists: https://www.youtube.com/watch?v=Eaz5e6M8tL4
[26] List methods: https://www.youtube.com/watch?v=8-RDVWGktuI
[27] Tuples: https://www.youtube.com/watch?v=bdS4dHIJGBc
[28] Tuple operations: https://www.youtube.com/watch?v=TItKabcTTQ4
[29] Dictionary: https://www.youtube.com/watch?v=4Q0pW8XBOkc
[30] Dictionary methods: https://www.youtube.com/watch?v=oLeNHuORpNY
[31] Files: https://www.youtube.com/watch?v=vuyb7CxZgbU
[32] https://www.youtube.com/watch?v=FqcjKewJTQ0
[33] File organization: https://www.youtube.com/watch?v=MRuq3SRXses
[34] OOP’s concepts: https://www.youtube.com/watch?v=qiSCMNBIP2g
[35] Inheritance: https://www.youtube.com/watch?v=Cn7AkDb4pIU
[36] Overriding: https://www.youtube.com/watch?v=CcTzTuIsoFk

Book Resources

For more supplementary concepts, programs, question bank, old question papers (VTU), quiz, recent trends in Python and educational resources, kindly visit the author's website at:

https://tocxten.com/

ABOUT THE AUTHOR

Dr. Thyagaraju G S is currently serving as a Professor and Head in the Department of Computer Science and Engineering at SDM Institute of Technology, Ujire. With over 20 years of teaching experience, he has authored and published more than 40 papers in esteemed peer-reviewed journals. Among these, over 10 papers have received approval from UGC and are indexed in Scopus Journals. His academic contributions extend to guiding the successful research journey of 1 scholar, while he is actively supervising the research of 3 scholars within the field of Computer Science under VTU. Dr. Thyagaraju G S boasts an impressive record of 11+ publications in international conferences and 2 books (1. “Programming in C and Introduction to Data Structures” 2016 (ISBN :9789385868665) 2. “Context Aware Computing for Ubiquitous Computing Applications”, published by LAP LAMBERT Academic Publishing, Germany, ISBN-13:978-3-659-68539, 2015). On the 20th of May, 2014, he earned a PhD degree from Visvesvaraya Technological University, Belagavi, specializing in the Faculty of Computer and Information Sciences. His doctoral dissertation titled “Context Aware Computing for Ubiquitous Computing Applications” focused on the design of algorithms and the development of context-sensitive recommendation systems. These systems utilized Artificial Intelligence methodologies such as Rule-Based approaches, Bayesian Networks, Rough Set Theory, and Decision Tables. His educational background encompasses a B.Sc, M.Sc in Physics, M.Tech in Computer Science, and a Ph.D in Computer Science. At present, Dr. Thyagaraju G S is deeply engrossed in research endeavors centered around Contextual and Conscious Quantum Artificial Intelligence, as well as Generative Artificial Intelligence.

ISBN: 978-93-5747-477-1

9 789357 474771

Selfypage Developers Pvt Ltd

MRP Rs. 590/-