# Python Basics: Comprehensive Guide

## Course Materials

### Introduction to Python

Python is a high-level, interpreted programming language known for its readability and simplicity. Created by Guido van Rossum and first released in 1991, Python has become one of the most popular programming languages in the world, used in web development, data science, artificial intelligence, automation, and many other fields.

#### Key Features of Python:

* **Readability**: Clean syntax that emphasizes readability
* **Versatility**: Can be used for a wide range of applications
* **Extensive Libraries**: Rich standard library and thousands of third-party packages
* **Interpreted**: No compilation needed
* **Dynamic Typing**: Variables are checked during runtime
* **Beginner-Friendly**: Easy to learn and use
* **Cross-Platform**: Runs on various operating systems

### Learning Objectives

By the end of this course, students will be able to: - Set up Python on their computer - Understand Python’s basic syntax - Work with various data types, operators, and functions - Use conditional statements - Implement basic error handling - Import and use modules

## 1. Setting Up Python

### Installing Python

Python can be downloaded from the official website: [python.org](https://python.org)

#### Installation Steps:

1. Download the latest version (Python 3.x recommended)
2. Run the installer
3. Check “Add Python to PATH” during installation
4. Verify installation by opening a command prompt/terminal and typing:

* python --version

### Integrated Development Environments (IDEs)

Popular Python IDEs and editors: - **PyCharm**: Full-featured IDE with advanced tools - **Visual Studio Code**: Lightweight editor with Python extensions - **Jupyter Notebook**: Browser-based interactive environment, ideal for data science - **IDLE**: Python’s built-in IDE (included with Python installation) - **Spyder**: Scientific-oriented IDE

### Your First Python Program

Create a file named hello.py with the following content:

print("Hello, World!")

Run it in your terminal or command prompt:

python hello.py

**Exercise 1**: Install Python, choose an IDE, and create a simple “Hello, World!” program.

## 2. Python Syntax Basics

### Indentation

Python uses indentation (whitespace) to define code blocks instead of curly braces.

**if** True:  
 print("This is indented with 4 spaces")  
 **if** True:  
 print("This is indented with 8 spaces")

**Rules**: - Standard indentation is 4 spaces (not tabs) - Consistent indentation is required - Incorrect indentation will cause errors

### Comments

Comments are used to explain code and are ignored by the Python interpreter.

*# This is a single-line comment*  
  
*"""*  
*This is a multi-line comment*  
*or docstring that spans multiple lines*  
*"""*

### Line Continuation

Long lines can be broken into multiple lines for readability:

*# Using backslash*  
total = 1 + 2 + 3 + \  
 4 + 5 + 6  
  
*# Using parentheses (preferred)*  
total = (1 + 2 + 3 +  
 4 + 5 + 6)

**Exercise 2**: Write a program with comments explaining each line and use proper indentation.

## 3. Variables and Data Types

### Variables

Variables are containers for storing data values. Python has no command for declaring variables - a variable is created when you first assign a value to it.

*# Variable assignment*  
name = "John"  
age = 30  
height = 5.9  
is\_student = True

**Variable Naming Rules**: - Can contain letters, numbers, and underscores - Cannot start with a number - Cannot use reserved words (e.g., if, for, while) - Case-sensitive (age and Age are different variables)

### Basic Data Types

Python has several built-in data types:

#### Numeric Types:

*# Integer*  
x = 10  
  
*# Float*  
y = 10.5  
  
*# Complex number*  
z = 1 + 2j

#### Text Type:

*# String*  
name = "Python"  
multiline\_string = """This is a  
multiline string"""

#### Boolean Type:

is\_valid = True  
is\_completed = False

#### None Type:

result = None *# Represents absence of value*

### Type Conversion

You can convert between different data types:

*# Explicit type conversion*  
x = int(10.8) *# x becomes 10*  
y = float(10) *# y becomes 10.0*  
z = str(10) *# z becomes "10"*  
b = bool(1) *# b becomes True*

### Checking Data Types

x = 10  
print(type(x)) *# <class 'int'>*

**Exercise 3**: Create variables of each data type and print their values and types.

## 4. Basic Operators

### Arithmetic Operators

*# Addition*  
result = 5 + 3 *# 8*  
  
*# Subtraction*  
result = 5 - 3 *# 2*  
  
*# Multiplication*  
result = 5 \* 3 *# 15*  
  
*# Division (returns float)*  
result = 5 / 2 *# 2.5*  
  
*# Floor division (returns integer)*  
result = 5 // 2 *# 2*  
  
*# Modulus (remainder)*  
result = 5 % 2 *# 1*  
  
*# Exponentiation*  
result = 2 \*\* 3 *# 8*

### Assignment Operators

x = 5 *# Basic assignment*  
x += 3 *# Equivalent to x = x + 3*  
x -= 2 *# Equivalent to x = x - 2*  
x \*= 4 *# Equivalent to x = x \* 4*  
x /= 2 *# Equivalent to x = x / 2*  
x //= 2 *# Equivalent to x = x // 2*  
x %= 2 *# Equivalent to x = x % 2*  
x \*\*= 2 *# Equivalent to x = x \*\* 2*

### Comparison Operators

x = 5  
y = 10  
  
x == y *# Equal to (False)*  
x != y *# Not equal to (True)*  
x > y *# Greater than (False)*  
x < y *# Less than (True)*  
x >= y *# Greater than or equal to (False)*  
x <= y *# Less than or equal to (True)*

### Logical Operators

x = True  
y = False  
  
x **and** y *# Logical AND (False)*  
x **or** y *# Logical OR (True)*  
**not** x *# Logical NOT (False)*

### Identity Operators

x = ["apple", "banana"]  
y = ["apple", "banana"]  
z = x  
  
x **is** z *# True (same object)*  
x **is** y *# False (different objects with same content)*  
x **is** **not** y *# True*

### Membership Operators

fruits = ["apple", "banana", "cherry"]  
  
*"banana"* **in** fruits *# True*  
*"orange"* **not** **in** fruits *# True*

**Exercise 4**: Create a calculator program that takes two numbers as input and performs all arithmetic operations on them.

## 5. Input and Output

### Input

The input() function allows user input:

name = input("Enter your name: ")  
print("Hello, " + name)  
  
*# Getting numeric input*  
age = int(input("Enter your age: "))

### Output

The print() function displays output:

*# Basic printing*  
print("Hello, World!")  
  
*# Printing multiple items*  
print("Name:", name, "Age:", age)  
  
*# Using formatted strings (f-strings) - Python 3.6+*  
print(f"Hello, {name}. You are {age} years old.")  
  
*# Using format() method*  
print("Hello, {}. You are {} years old.".format(name, age))  
  
*# Using % operator (older style)*  
print("Hello, %s. You are %d years old." % (name, age))

#### Print options:

*# Changing separator*  
print("apple", "banana", "cherry", sep=", ")  
  
*# Changing end character (default is newline)*  
print("Hello", end=" ")  
print("World") *# Prints "Hello World" on the same line*

**Exercise 5**: Create a program that asks for the user’s name, age, and favorite color, then prints a formatted message using their input.

## 6. String Manipulation

Strings are sequences of characters enclosed in quotes.

### String Operations

*# Concatenation*  
first\_name = "John"  
last\_name = "Doe"  
full\_name = first\_name + " " + last\_name *# "John Doe"*  
  
*# Repetition*  
line = "-" \* 20 *# "--------------------"*  
  
*# Indexing (starts at 0)*  
message = "Hello"  
first\_char = message[0] *# "H"*  
  
*# Negative indexing (starts from the end)*  
last\_char = message[-1] *# "o"*  
  
*# Slicing*  
message = "Hello, World!"  
substring = message[0:5] *# "Hello"*  
substring = message[:5] *# "Hello" (from beginning to index 4)*  
substring = message[7:] *# "World!" (from index 7 to end)*  
substring = message[-6:] *# "World!" (last 6 characters)*

### String Methods

message = "Hello, World!"  
  
*# Case conversion*  
upper\_message = message.upper() *# "HELLO, WORLD!"*  
lower\_message = message.lower() *# "hello, world!"*  
title\_message = message.title() *# "Hello, World!"*  
  
*# Finding substrings*  
position = message.find("World") *# 7 (returns -1 if not found)*  
count = message.count("l") *# 3*  
  
*# Replacing substrings*  
new\_message = message.replace("World", "Python") *# "Hello, Python!"*  
  
*# Checking string properties*  
is\_alpha = "Hello".isalpha() *# True (only alphabetic characters)*  
is\_digit = "123".isdigit() *# True (only digits)*  
is\_alnum = "Hello123".isalnum() *# True (alphabetic or numeric)*  
starts\_with = message.startswith("Hello") *# True*  
ends\_with = message.endswith("!") *# True*  
  
*# Removing whitespace*  
text = " Hello "  
stripped = text.strip() *# "Hello"*  
left\_stripped = text.lstrip() *# "Hello "*  
right\_stripped = text.rstrip() *# " Hello"*  
  
*# Splitting and joining*  
words = message.split(", ") *# ["Hello", "World!"]*  
joined = ", ".join(["Hello", "Python"]) *# "Hello, Python"*

**Exercise 6**: Create a program that takes a sentence from the user, counts how many words it contains, and prints each word in reverse order.

## 7. Lists

Lists are ordered, mutable collections of items that can store different data types.

### Creating Lists

*# Empty list*  
empty\_list = []  
empty\_list = list()  
  
*# List with items*  
fruits = ["apple", "banana", "cherry"]  
mixed\_list = [1, "Hello", 3.14, True]

### Accessing List Elements

fruits = ["apple", "banana", "cherry", "orange", "kiwi"]  
  
*# Indexing*  
first\_fruit = fruits[0] *# "apple"*  
last\_fruit = fruits[-1] *# "kiwi"*  
  
*# Slicing*  
some\_fruits = fruits[1:3] *# ["banana", "cherry"]*

### Modifying Lists

fruits = ["apple", "banana", "cherry"]  
  
*# Changing an element*  
fruits[1] = "blackcurrant" *# ["apple", "blackcurrant", "cherry"]*  
  
*# Adding elements*  
fruits.append("orange") *# Adds to the end*  
fruits.insert(1, "mango") *# Inserts at a specific position*  
fruits.extend(["kiwi", "lemon"]) *# Adds multiple items*  
  
*# Removing elements*  
fruits.remove("banana") *# Removes a specific item*  
popped\_fruit = fruits.pop() *# Removes and returns the last item*  
popped\_fruit = fruits.pop(1) *# Removes and returns item at index 1*  
**del** fruits[0] *# Deletes item at index 0*  
fruits.clear() *# Empties the list*

### List Operations

*# List concatenation*  
list1 = [1, 2, 3]  
list2 = [4, 5, 6]  
combined = list1 + list2 *# [1, 2, 3, 4, 5, 6]*  
  
*# List repetition*  
repeated = list1 \* 3 *# [1, 2, 3, 1, 2, 3, 1, 2, 3]*  
  
*# List methods*  
numbers = [3, 1, 4, 1, 5, 9, 2]  
numbers.sort() *# [1, 1, 2, 3, 4, 5, 9]*  
numbers.reverse() *# [9, 5, 4, 3, 2, 1, 1]*  
count = numbers.count(1) *# 2 (count of value 1)*  
index = numbers.index(5) *# 2 (first occurrence of value 5)*  
  
*# Finding list length*  
length = len(numbers) *# 7*  
  
*# Checking membership*  
contains = 5 **in** numbers *# True*

### List Comprehensions

A concise way to create lists based on existing lists:

*# Create a list of squares*  
squares = [x\*\*2 **for** x **in** range(10)] *# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]*  
  
*# List comprehension with condition*  
even\_squares = [x\*\*2 **for** x **in** range(10) **if** x % 2 == 0] *# [0, 4, 16, 36, 64]*

**Exercise 7**: Create a program that asks the user for a list of numbers, then calculates and displays the sum, average, maximum, and minimum values.

## 8. Tuples

Tuples are ordered, immutable collections of items that can store different data types.

### Creating Tuples

*# Empty tuple*  
empty\_tuple = ()  
empty\_tuple = tuple()  
  
*# Tuple with items*  
fruits = ("apple", "banana", "cherry")  
mixed\_tuple = (1, "Hello", 3.14)  
  
*# Single-item tuple (note the comma)*  
single\_item = ("apple",)

### Accessing Tuple Elements

fruits = ("apple", "banana", "cherry", "orange", "kiwi")  
  
*# Indexing*  
first\_fruit = fruits[0] *# "apple"*  
last\_fruit = fruits[-1] *# "kiwi"*  
  
*# Slicing*  
some\_fruits = fruits[1:3] *# ("banana", "cherry")*

### Tuple Operations

*# Cannot modify tuples (immutable)*  
*# fruits[0] = "pear" # This would cause an error*  
  
*# Tuple concatenation*  
tuple1 = (1, 2, 3)  
tuple2 = (4, 5, 6)  
combined = tuple1 + tuple2 *# (1, 2, 3, 4, 5, 6)*  
  
*# Tuple repetition*  
repeated = tuple1 \* 3 *# (1, 2, 3, 1, 2, 3, 1, 2, 3)*  
  
*# Tuple methods*  
numbers = (3, 1, 4, 1, 5)  
count = numbers.count(1) *# 2 (count of value 1)*  
index = numbers.index(5) *# 4 (first occurrence of value 5)*  
  
*# Finding tuple length*  
length = len(numbers) *# 5*  
  
*# Checking membership*  
contains = 5 **in** numbers *# True*

### Why Use Tuples?

* Tuples are faster than lists
* Protect data that shouldn’t change
* Can be used as dictionary keys (unlike lists)
* For returning multiple values from a function

**Exercise 8**: Create a program that uses a tuple to store the names of the days of the week, then asks the user for a number (1-7) and returns the corresponding day.

## 9. Dictionaries

Dictionaries are unordered, mutable collections of key-value pairs.

### Creating Dictionaries

*# Empty dictionary*  
empty\_dict = {}  
empty\_dict = dict()  
  
*# Dictionary with items*  
person = {  
 "name": "John",  
 "age": 30,  
 "city": "New York"  
}  
  
*# Alternative creation method*  
person = dict(name="John", age=30, city="New York")

### Accessing Dictionary Elements

person = {"name": "John", "age": 30, "city": "New York"}  
  
*# Accessing values using keys*  
name = person["name"] *# "John"*  
  
*# Using get() method (safer, returns None if key doesn't exist)*  
age = person.get("age") *# 30*  
country = person.get("country") *# None*  
country = person.get("country", "Unknown") *# "Unknown" (default value)*

### Modifying Dictionaries

person = {"name": "John", "age": 30, "city": "New York"}  
  
*# Adding or changing items*  
person["email"] = "john@example.com" *# Adds new key-value pair*  
person["age"] = 31 *# Updates existing value*  
  
*# Removing items*  
removed\_value = person.pop("city") *# Removes and returns value*  
last\_item = person.popitem() *# Removes and returns last item as tuple*  
**del** person["age"] *# Removes key-value pair*  
person.clear() *# Empties the dictionary*

### Dictionary Methods

person = {"name": "John", "age": 30, "city": "New York"}  
  
*# Getting all keys*  
keys = person.keys() *# dict\_keys(['name', 'age', 'city'])*  
  
*# Getting all values*  
values = person.values() *# dict\_values(['John', 30, 'New York'])*  
  
*# Getting all key-value pairs*  
items = person.items() *# dict\_items([('name', 'John'), ('age', 30), ('city', 'New York')])*  
  
*# Updating dictionary with another dictionary*  
person.update({"age": 31, "country": "USA"})  
  
*# Creating a copy*  
person\_copy = person.copy()

### Dictionary Comprehensions

*# Create a dictionary of squares*  
squares = {x: x\*\*2 **for** x **in** range(5)} *# {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}*  
  
*# Dictionary comprehension with condition*  
even\_squares = {x: x\*\*2 **for** x **in** range(10) **if** x % 2 == 0}

**Exercise 9**: Create a simple address book program that allows users to add, view, and delete contacts (name and phone number) using a dictionary.

## 10. Sets

Sets are unordered, mutable collections of unique items.

### Creating Sets

*# Empty set*  
empty\_set = set() *# Note: {} creates an empty dictionary, not a set*  
  
*# Set with items*  
fruits = {"apple", "banana", "cherry"}  
mixed\_set = {1, "Hello", 3.14}

### Set Operations

fruits = {"apple", "banana", "cherry"}  
  
*# Adding elements*  
fruits.add("orange") *# Adds a single element*  
fruits.update(["mango", "kiwi"]) *# Adds multiple elements*  
  
*# Removing elements*  
fruits.remove("banana") *# Removes element (raises error if not found)*  
fruits.discard("pear") *# Removes element (no error if not found)*  
popped\_item = fruits.pop() *# Removes and returns an arbitrary element*  
fruits.clear() *# Empties the set*  
  
*# Set operations*  
set1 = {1, 2, 3, 4, 5}  
set2 = {4, 5, 6, 7, 8}  
  
union = set1 | set2 *# {1, 2, 3, 4, 5, 6, 7, 8}*  
intersection = set1 & set2 *# {4, 5}*  
difference = set1 - set2 *# {1, 2, 3}*  
symmetric\_diff = set1 ^ set2 *# {1, 2, 3, 6, 7, 8}*  
  
*# Checking membership*  
contains = "apple" **in** fruits *# True*  
  
*# Set size*  
size = len(fruits) *# Number of elements in the set*

### Set Comprehensions

*# Create a set of squares*  
squares = {x\*\*2 **for** x **in** range(10)} *# {0, 1, 64, 4, 36, 9, 16, 49, 81, 25}*  
  
*# Set comprehension with condition*  
even\_squares = {x\*\*2 **for** x **in** range(10) **if** x % 2 == 0}

**Exercise 10**: Create a program that takes two lists of numbers from the user, converts them to sets, and shows their union, intersection, and difference.

## 11. Conditional Statements

Conditional statements allow you to execute certain code blocks based on conditions.

### if Statement

age = 18  
  
**if** age >= 18:  
 print("You are an adult")

### if-else Statement

age = 16  
  
**if** age >= 18:  
 print("You are an adult")  
**else**:  
 print("You are a minor")

### if-elif-else Statement

score = 85  
  
**if** score >= 90:  
 grade = "A"  
**elif** score >= 80:  
 grade = "B"  
**elif** score >= 70:  
 grade = "C"  
**elif** score >= 60:  
 grade = "D"  
**else**:  
 grade = "F"  
  
print(f"Your grade is {grade}")

### Nested Conditionals

age = 25  
has\_license = True  
  
**if** age >= 18:  
 **if** has\_license:  
 print("You can drive")  
 **else**:  
 print("You need to get a license first")  
**else**:  
 print("You are too young to drive")

### Conditional Expressions (Ternary Operator)

age = 20  
status = "adult" **if** age >= 18 **else** "minor"

### Using Logical Operators in Conditions

age = 25  
income = 50000  
  
**if** age > 18 **and** income > 30000:  
 print("You qualify for a loan")  
  
**if** age < 18 **or** income < 30000:  
 print("You don't qualify for a loan")  
  
**if** **not** has\_criminal\_record:  
 print("Background check passed")

**Exercise 11**: Create a BMI calculator that takes a person’s weight (in kg) and height (in m), then categorizes them as underweight, normal, overweight, or obese.

## Assessment Ideas

1. **Multiple Choice Quiz**: Test basic understanding of Python concepts.
2. **Code Debugging**: Provide code with errors and have students fix it.
3. **Code Completion**: Give partial code and have students complete it.
4. **Mini-Project**: Implement a small application using the concepts learned.
5. **Pair Programming**: Have students work in pairs to solve a coding problem.

## Additional Resources

* Python Documentation: <https://docs.python.org/3/>
* Python Standard Library: <https://docs.python.org/3/library/>
* Book: “Python Crash Course” by Eric Matthes
* Book: “Automate the Boring Stuff with Python” by Al Sweigart
* Online Course: “Python for Everybody” by Dr. Charles Severance
* Interactive Learning: <https://www.codecademy.com/learn/learn-python-3>

## Conclusion

This comprehensive guide covers the fundamentals of Python programming up to loops. By mastering these basics, you’ll have a solid foundation to build upon as you explore more advanced Python concepts and applications. Remember that programming is a skill that improves with practice, so try to work through the exercises and build your own small projects.