Python Programming

Beginner's Complete Guide

Comprehensive Notes with Code Examples

1. Python Outputs

Python outputs refer to displaying information to the user or console. The primary function used for output in Python is the print() function, which displays data on the screen.

1.1 Basic print() Function

The print() function is used to display output to the console. It can print text, numbers, variables, and expressions.

```
# Example 1: Simple text output
print('Hello, World!')

# Example 2: Printing numbers
print(42)

# Example 3: Printing floating-point numbers
print(3.14159)

# Example 4: Printing multiple values
print('Python', 'is', 'awesome')

# Example 5: Printing with variables
name = 'Alice'
print(name)

# Example 6: Printing mathematical expressions
print(10 + 20)

# Example 7: Printing boolean values
print(True)

# Example 8: Printing with concatenation
print('My age is ' + str(25))

# Example 9: Empty print (new line)
print()

# Example 10: Printing special characters
print('Hello\nWorld')

# Example 11: Printing tab character
print('Name:\tJohn')

# Example 12: Printing quotes
print("He said, 'Python is fun!'")

# Example 13: Printing with escape characters
print('C:\\Users\\Documents')

# Example 14: Printing unicode characters
print('\u2665 Heart symbol')

# Example 15: Printing raw strings
print(r'C:\new\folder')
```

1.2 print() with sep Parameter

The sep parameter specifies the separator between multiple values in print(). By default, it's a space.

```
# Example 1: Default separator (space)
print('Apple', 'Banana', 'Cherry')

# Example 2: Comma separator
print('Apple', 'Banana', 'Cherry', sep=', ')

# Example 3: Dash separator
print('2024', '01', '15', sep='-')
```

```
# Example 4: No separator
print('Hello', 'World', sep='')

# Example 5: Custom separator with symbol
print('Python', 'Java', 'C++', sep=' | ')

# Example 6: Newline as separator
print('First', 'Second', 'Third', sep='\n')

# Example 7: Multiple characters separator
print('One', 'Two', 'Three', sep=' -> ')

# Example 8: Tab separator
print('Name', 'Age', 'City', sep='\t')

# Example 9: Asterisk separator
print(1, 2, 3, 4, 5, sep=' * ')

# Example 10: Colon separator for time format
print(10, 30, 45, sep=':')

# Example 11: Underscore separator
print('first', 'name', sep='_')

# Example 12: Slash separator for dates
print(15, 8, 2024, sep='/')

# Example 13: Multiple variables with custom sep
x, y, z = 1, 2, 3
print(x, y, z, sep=' + ')

# Example 14: Combining with calculations
print('Sum:', 10 + 20, sep=' = ')

# Example 15: Empty string separator
print('a', 'b', 'c', sep='')
```

1.3 print() with end Parameter

The end parameter specifies what to print at the end. By default, it's a newline character (\n).

```
# Example 1: Default end (newline)
print('Hello')
print('World')

# Example 2: Space at the end
print('Hello', end=' ')
print('World')

# Example 3: No newline
print('Python', end='')
print('Programming')

# Example 4: Custom ending
print('Loading', end='...')
print('Done')

# Example 5: Multiple prints in one line
print('A', end=' ')
print('B', end=' ')
print('C')

# Example 6: Tab at the end
print('Name:', end='\t')
print('John')

# Example 7: Creating progress indicator
for i in range(3):
    print('*', end='')

# Example 8: Dash ending
print('Start', end=' - ')
print('End')
```

```
# Example 9: Exclamation mark ending
print('Warning', end='!!!')
print()

# Example 10: Arrow ending
print('Step 1', end=' -> ')
print('Step 2')

# Example 11: Colon ending
print('Result', end=': ')
print(100)

# Example 12: Comma ending
print('Item 1', end=', ')
print('Item 2')

# Example 13: Custom message ending
print('Error', end=' occurred\n')
print('Please try again')

# Example 14: Combining sep and end
print('A', 'B', 'C', sep='-', end='!\n')

# Example 15: Empty string ending
print('Python', end='')
print('3.x')
```

2. Python Data Types

Data types define the type of data that can be stored in a variable. Python has several built-in data types including numeric types (int, float, complex), sequence types (str, list, tuple), mapping type (dict), set types (set, frozenset), and boolean type (bool).

2.1 Integer (int) Data Type

Integers are whole numbers without decimal points. They can be positive, negative, or zero. Python integers have unlimited precision.

```
# Example 1: Positive integer
x = 42
print(x, type(x))

# Example 2: Negative integer
y = -15
print(y, type(y))

# Example 3: Zero
z = 0
print(z, type(z))

# Example 4: Large integer
big_num = 12345678901234567890123456789O
print(big_num)

# Example 5: Integer arithmetic
result = 10 + 20
print(result)

# Example 6: Integer multiplication
product = 7 * 8
print(product)

# Example 7: Integer division (floor)
quotient = 17 // 3
print(quotient)

# Example 8: Integer power
power = 2 ** 10
print(power)

# Example 9: Integer from user input
age = int(input('Enter age: '))

# Example 10: Checking type
num = 100
print(isinstance(num, int))

# Example 11: Binary integer
binary_num = 0b1010
print(binary_num)

# Example 12: Octal integer
octal_num = 0o17
print(octal_num)

# Example 13: Hexadecimal integer
hex_num = 0xFF
print(hex_num)

# Example 14: Integer modulo operation
remainder = 17 % 5
print(remainder)

# Example 15: Absolute value of integer
abs_value = abs(-25)
print(abs_value)
```

2.2 Float Data Type

Floats are numbers with decimal points or exponential notation. They represent real numbers and are useful for precise calculations.

```
# Example 1: Simple float
x = 3.14
print(x, type(x))

# Example 2: Negative float
y = -2.5
print(y, type(y))

# Example 3: Float with many decimals
pi = 3.141592653589793
print(pi)

# Example 4: Scientific notation
sci_num = 2.5e3
print(sci_num)

# Example 5: Small scientific notation
small = 1.5e-4
print(small)

# Example 6: Float division
result = 10 / 3
print(result)

# Example 7: Float from integer
float_num = float(10)
print(float_num)

# Example 8: Adding floats
sum_float = 1.5 + 2.7
print(sum_float)

# Example 9: Float multiplication
product = 3.14 * 2
print(product)

# Example 10: Rounding floats
rounded = round(3.14159, 2)
print(rounded)

# Example 11: Float comparison
print(3.0 == 3)

# Example 12: Float to int conversion
int_from_float = int(5.9)
print(int_from_float)

# Example 13: Checking float type
num = 2.5
print(isinstance(num, float))

# Example 14: Float with zero
zero_float = 0.0
print(zero_float, type(zero_float))

# Example 15: Float infinity
infinity = float('inf')
print(infinity)
```

2.3 String (str) Data Type

Strings are sequences of characters enclosed in quotes (single, double, or triple). They are immutable and can contain letters, numbers, and special characters.

```
# Example 1: Single quotes string
name = 'Python'
print(name, type(name))
```

```
# Example 2: Double quotes string
message = "Hello World"
print(message)

# Example 3: Triple quotes string
long_text = '''This is
a multi-line
string'''
print(long_text)

# Example 4: Empty string
empty = ''
print(len(empty))

# Example 5: String concatenation
full_name = 'John' + ' ' + 'Doe'
print(full_name)

# Example 6: String repetition
repeated = 'Ha' * 3
print(repeated)

# Example 7: String indexing
word = 'Python'
print(word[0])

# Example 8: String slicing
text = 'Hello World'
print(text[0:5])

# Example 9: String length
length = len('Python')
print(length)

# Example 10: String uppercase
uppercase = 'hello'.upper()
print(uppercase)

# Example 11: String lowercase
lowercase = 'PYTHON'.lower()
print(lowercase)

# Example 12: String replace
new_str = 'Hello World'.replace('World', 'Python')
print(new_str)

# Example 13: String split
words = 'Python is awesome'.split()
print(words)

# Example 14: String formatting
name = 'Alice'
print(f'Hello, {name}!')

# Example 15: String contains check
result = 'Python' in 'I love Python'
print(result)
```

2.4 Boolean (bool) Data Type

Boolean represents one of two values: True or False. Booleans are used in conditional statements and logical operations.

```
# Example 1: Boolean True
is_active = True
print(is_active, type(is_active))

# Example 2: Boolean False
is_finished = False
print(is_finished)

# Example 3: Boolean from comparison
result = 5 > 3
print(result)

# Example 4: Boolean from equality
is_equal = (10 == 10)
print(is_equal)

# Example 5: Boolean NOT operation
value = not True
print(value)

# Example 6: Boolean AND operation
result = True and False
print(result)

# Example 7: Boolean OR operation
result = True or False
print(result)

# Example 8: Boolean from integer
bool_val = bool(1)
print(bool_val)

# Example 9: Boolean from zero
bool_zero = bool(0)
print(bool_zero)

# Example 10: Boolean from string
bool_str = bool('Python')
print(bool_str)

# Example 11: Boolean from empty string
bool_empty = bool('')
print(bool_empty)

# Example 12: Boolean in condition
if True:
    print('This will execute')

# Example 13: Boolean with isinstance
print(isinstance(True, bool))

# Example 14: Boolean from comparison chain
result = 1 < 2 < 3
print(result)

# Example 15: Boolean negation
is_valid = False
print(not is_valid)
```

2.5 List Data Type

Lists are ordered, mutable collections that can contain items of different types. Lists are defined using square brackets [].

```
# Example 1: Simple list
fruits = ['apple', 'banana', 'cherry']
print(fruits, type(fruits))
```

```
# Example 2: Empty list
empty_list = []
print(empty_list)

# Example 3: List with different types
mixed = [1, 'hello', 3.14, True]
print(mixed)

# Example 4: List indexing
numbers = [10, 20, 30, 40]
print(numbers[0])

# Example 5: List slicing
print(numbers[1:3])

# Example 6: List append
fruits = ['apple']
fruits.append('banana')
print(fruits)

# Example 7: List extend
list1 = [1, 2]
list1.extend([3, 4])
print(list1)

# Example 8: List insert
colors = ['red', 'blue']
colors.insert(1, 'green')
print(colors)

# Example 9: List remove
fruits = ['apple', 'banana', 'cherry']
fruits.remove('banana')
print(fruits)

# Example 10: List pop
numbers = [1, 2, 3, 4]
last = numbers.pop()
print(numbers, last)

# Example 11: List length
my_list = [1, 2, 3, 4, 5]
print(len(my_list))

# Example 12: List sorting
nums = [3, 1, 4, 1, 5]
nums.sort()
print(nums)

# Example 13: List reverse
letters = ['a', 'b', 'c']
letters.reverse()
print(letters)

# Example 14: List count
numbers = [1, 2, 2, 3, 2]
print(numbers.count(2))

# Example 15: List comprehension
squares = [x**2 for x in range(5)]
print(squares)
```

3. Variables

Variables are containers for storing data values. In Python, variables are created when you assign a value to them. Python is dynamically typed, meaning you don't need to declare the variable type explicitly.

3.1 Variable Assignment

Variables are created using the assignment operator (=). The variable name goes on the left, and the value goes on the right.

```
# Example 1: Simple integer assignment
x = 10
print(x)

# Example 2: String assignment
name = 'Alice'
print(name)

# Example 3: Float assignment
pi = 3.14159
print(pi)

# Example 4: Boolean assignment
is_student = True
print(is_student)

# Example 5: Multiple assignment
a = b = c = 5
print(a, b, c)

# Example 6: Multiple variables
x, y, z = 1, 2, 3
print(x, y, z)

# Example 7: Swapping variables
a, b = 10, 20
a, b = b, a
print(a, b)

# Example 8: Assigning from expression
result = 10 + 20
print(result)

# Example 9: Reassigning variable
count = 5
count = 10
print(count)

# Example 10: Variable from another variable
x = 100
y = x
print(y)

# Example 11: List assignment
fruits = ['apple', 'banana']
print(fruits)

# Example 12: Dictionary assignment
person = {'name': 'John', 'age': 25}
print(person)

# Example 13: Tuple assignment
coordinates = (10, 20)
print(coordinates)

# Example 14: None assignment
value = None
print(value)

# Example 15: Assignment from function
length = len('Python')
print(length)
```

3.2 Variable Naming Rules

Variable names must follow certain rules: they must start with a letter or underscore, can contain letters, numbers, and underscores, are case-sensitive, and cannot be Python keywords.

```
# Example 1: Valid name with letters
student_name = 'Bob'
print(student_name)

# Example 2: Valid name with underscore
_count = 5
print(_count)

# Example 3: Valid name with numbers
value1 = 100
print(value1)

# Example 4: Camel case naming
studentAge = 20
print(studentAge)

# Example 5: Snake case naming (recommended)
first_name = 'John'
print(first_name)

# Example 6: All uppercase (constants)
MAX_SIZE = 1000
print(MAX_SIZE)

# Example 7: Case sensitivity
age = 25
Age = 30
print(age, Age)

# Example 8: Starting with underscore
_private = 'hidden'
print(_private)

# Example 9: Double underscore
__special = 42
print(__special)

# Example 10: Descriptive naming
total_students = 50
print(total_students)

# Example 11: Multiple word names
user_email_address = 'user@example.com'
print(user_email_address)

# Example 12: Single letter (avoid in real code)
x = 10
print(x)

# Example 13: Alphanumeric combination
test123 = 'value'
print(test123)

# Example 14: Mixed case
MyVariable = 'test'
print(MyVariable)

# Example 15: Meaningful names
student_count = 30
print(student_count)
```

3.3 Variable Scope

Variable scope determines where a variable can be accessed. Python has local scope (inside functions) and global scope (outside functions).

```
# Example 1: Global variable
x = 10
print(x)
```

```
# Example 2: Local variable in function
def my_func():
    y = 20
    print(y)
my_func()
```

```
# Example 3: Accessing global in function
count = 5
def show():
    print(count)
show()
```

```
# Example 4: Global keyword
total = 0
def add():
    global total
    total += 10
add()
print(total)
```

```
# Example 5: Local shadows global
x = 100
def func():
    x = 50
    print(x)
func()
print(x)
```

```
# Example 6: Nested function scope
def outer():
    x = 10
    def inner():
        print(x)
    inner()
outer()
```

```
# Example 7: nonlocal keyword
def outer():
    x = 5
    def inner():
        nonlocal x
        x = 10
    inner()
    print(x)
outer()
```

```
# Example 8: Global list modification
my_list = []
def add_item():
    my_list.append(1)
add_item()
print(my_list)
```

```
# Example 9: Multiple global variables
a, b = 1, 2
def show():
    print(a, b)
show()
```

```
# Example 10: Local variable priority
value = 'global'
def func():
    value = 'local'
    print(value)
func()
```

```
# Example 11: Return from function
def get_value():
    result = 100
    return result
num = get_value()
print(num)
```

```
# Example 12: Class variable scope
class MyClass:
    class_var = 10
print(MyClass.class_var)
```

```
# Example 13: Instance variable
class Person:
    def __init__(self):
        self.name = 'John'
p = Person()
print(p.name)
```

```
# Example 14: Built-in scope
print(len([1, 2, 3]))
```

```
# Example 15: Del statement
x = 10
del x
# print(x) would raise error
```

4. Comments in Python

Comments are used to explain code and make it more readable. Python ignores comments when executing code. They help developers understand the purpose and functionality of code sections.

4.1 Single-Line Comments

Single-line comments start with the hash symbol (#). Everything after # on that line is ignored by Python.

```
# Example 1: Basic comment
# This is a comment
print('Hello')

# Example 2: Inline comment
x = 10  # This assigns 10 to x

# Example 3: Multiple single-line comments
# First comment
# Second comment
print('Hi')

# Example 4: Comment explaining code
# Calculate area of rectangle
area = 5 * 10

# Example 5: Temporary code removal
# print('This will not execute')
print('This will')

# Example 6: Comment with special characters
# TODO: Fix this bug!
print('Test')

# Example 7: Comment for variable explanation
PI = 3.14159  # Mathematical constant

# Example 8: Comment with calculations
result = 100 + 200  # Sum of two numbers

# Example 9: Comment at start of line
# Initialize counter
counter = 0

# Example 10: Comment for debugging
print('Debug')  # Remove this later

# Example 11: Comment with date
# Created on: 2024-01-15
print('Hello')

# Example 12: Comment for author
# Author: John Doe
name = 'John'

# Example 13: Comment for version
# Version 1.0
print('Program started')

# Example 14: Comment with URL
# Reference: https://python.org
print('Test')

# Example 15: Empty comment
#
print('After empty comment')
```

4.2 Multi-Line Comments

Multi-line comments can be created using triple quotes (''' or """). Although technically string literals, they serve as comments when not assigned to variables.

```
# Example 1: Triple single quotes
'''
This is a
multi-line comment
'''
print('Hello')
```

```
# Example 2: Triple double quotes
"""
Another multi-line
comment example
"""
print('Hi')
```

```
# Example 3: Function documentation
def add(a, b):
    '''
    Adds two numbers
    '''
    return a + b
```

```
# Example 4: Class documentation
class Dog:
    '''
    Represents a dog
    '''
    pass
```

```
# Example 5: Module documentation
'''
Module: calculator.py
Purpose: Math operations
'''
```

```
# Example 6: Long explanation
'''
This program calculates
the factorial of a number
using recursion
'''
```

```
# Example 7: Code block comment
'''
The following code
handles user input
and validation
'''
x = 10
```

```
# Example 8: Inline multi-line
result = 5 * 10  '''
Calculates area
'''
```

```
# Example 9: Commenting out code block
'''
print('Line 1')
print('Line 2')
print('Line 3')
'''
```

```
# Example 10: Detailed explanation
'''
Algorithm:
1. Initialize variables
2. Process data
3. Return result
'''
```

```
# Example 11: Parameter documentation
def greet(name):
    '''
    name: str - person's name
    '''
    print(f'Hello {name}')
```

```
# Example 12: Return documentation
def square(x):
    '''
    Returns: x squared
    '''
    return x ** 2
```

```
# Example 13: Example in docstring
def multiply(a, b):
    '''
    Example: multiply(3, 4) = 12
    '''
    return a * b
```

```
# Example 14: Warning in comment
'''
WARNING: Do not modify
this section without testing
'''
```

```
# Example 15: License comment
'''
Copyright (c) 2024
Licensed under MIT
'''
```

4.3 Best Practices for Comments

Good comments explain why code does something, not what it does (which should be obvious from the code itself). They should be clear, concise, and kept up-to-date with code changes.

```
# Example 1: Good comment (explains why)
# Using binary search for performance
result = binary_search(data, target)
```

```
# Example 2: Bad comment (states obvious)
# Add 1 to x
x = x + 1
```

```
# Example 3: TODO comment
# TODO: Optimize this function
def slow_function():
    pass
```

```
# Example 4: FIXME comment
# FIXME: Handle edge case
def process(data):
    pass
```

```
# Example 5: NOTE comment
# NOTE: This assumes sorted input
def find_median(numbers):
    pass
```

```
# Example 6: HACK comment
# HACK: Temporary workaround
value = str(data) + '0'
```

```
# Example 7: Explaining complex logic
# Calculate using quadratic formula
result = (-b + sqrt(b**2 - 4*a*c)) / (2*a)
```

```
# Example 8: Business logic explanation
# Apply 10% discount for members
price = base_price * 0.9
```

```
# Example 9: Warning comment
# WARNING: This modifies the original list
data.sort()
```

```
# Example 10: Performance note
# O(n) time complexity
for item in collection:
    process(item)
```

```
# Example 11: Dependency note
# Requires: requests library
import requests
```

```
# Example 12: Assumption comment
# Assumes: input is positive
def sqrt(n):
    return n ** 0.5
```

```
# Example 13: Reference comment
# Based on: Algorithm from CLRS book
def quicksort(arr):
    pass
```

```
# Example 14: Deprecation warning
# DEPRECATED: Use new_function() instead
def old_function():
    pass
```

```
# Example 15: Section separator
# ===== Data Processing Section =====
data = process_data()
```

5. Keywords & Identifiers

Keywords are reserved words in Python that have special meanings and cannot be used as identifiers. Identifiers are names given to variables, functions, classes, etc. Understanding these is crucial for writing correct Python code.

5.1 Python Keywords

Python has 35 keywords (as of Python 3.x) that are reserved and have specific purposes. These include: False, None, True, and, as, assert, async, await, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield.

```
# Example 1: if keyword
if 5 > 3:
    print('True')
```

```
# Example 2: else keyword
if False:
    print('No')
else:
    print('Yes')
```

```
# Example 3: elif keyword
x = 10
if x < 5:
    print('Small')
elif x < 15:
    print('Medium')
```

```
# Example 4: for keyword
for i in range(3):
    print(i)
```

```
# Example 5: while keyword
count = 0
while count < 3:
    print(count)
    count += 1
```

```
# Example 6: break keyword
for i in range(5):
    if i == 3:
        break
    print(i)
```

```
# Example 7: continue keyword
for i in range(5):
    if i == 2:
        continue
    print(i)
```

```
# Example 8: def keyword
def my_function():
    print('Hello')
my_function()
```

```
# Example 9: return keyword
def add(a, b):
    return a + b
result = add(3, 4)
```

```
# Example 10: class keyword
class MyClass:
    pass
obj = MyClass()
```

```
# Example 11: import keyword
import math
print(math.pi)
```

```
# Example 12: from keyword
from math import sqrt
print(sqrt(16))
```

```
# Example 13: try-except keywords
try:
    x = 10 / 0
except:
    print('Error')
```

```
# Example 14: pass keyword
def empty_function():
    pass
```

```
# Example 15: in keyword
fruits = ['apple', 'banana']
print('apple' in fruits)
```

5.2 Valid Identifiers

Identifiers are names used to identify variables, functions, classes, modules, or other objects. They must start with a letter (a-z, A-Z) or underscore (_), followed by letters, digits (0-9), or underscores. They are case-sensitive and cannot be keywords.

```
# Example 1: Valid identifier - variable
student_name = 'Alice'
print(student_name)
```

```
# Example 2: Valid identifier - function
def calculate_sum():
    return 10 + 20
print(calculate_sum())
```

```
# Example 3: Valid identifier - class
class StudentRecord:
    pass
student = StudentRecord()
```

```
# Example 4: Identifier with numbers
value1 = 100
value2 = 200
print(value1, value2)
```

```
# Example 5: Identifier starting with underscore
_private_var = 42
print(_private_var)
```

```
# Example 6: All uppercase identifier (constant)
MAX_SIZE = 1000
MIN_SIZE = 10
print(MAX_SIZE)
```

```
# Example 7: Camel case identifier
userEmailAddress = 'user@example.com'
print(userEmailAddress)
```

```
# Example 8: Snake case identifier
first_name = 'John'
last_name = 'Doe'
print(first_name, last_name)
```

```
# Example 9: Single letter identifier
x = 10
y = 20
z = x + y
print(z)
```

```
# Example 10: Long descriptive identifier
total_number_of_students = 50
print(total_number_of_students)
```

```
# Example 11: Identifier with multiple underscores
__special_value__ = 100
print(__special_value__)
```

```
# Example 12: Class name convention
class BankAccount:
    def __init__(self):
        self.balance = 0
```

```
account = BankAccount()
```

```
# Example 13: Method identifier
class Calculator:
    def add_numbers(self, a, b):
        return a + b
calc = Calculator()
print(calc.add_numbers(5, 3))
```

```
# Example 14: Module-level identifier
_module_version = '1.0'
print(_module_version)
```

```
# Example 15: Unicode identifier (advanced)
π = 3.14159
print(π)
```

5.3 Naming Conventions

Python follows PEP 8 naming conventions: variables and functions use snake_case, classes use PascalCase, constants use UPPER_CASE, and 'private' attributes start with underscore. Following these conventions makes code more readable and maintainable.

```
# Example 1: Variable naming (snake_case)
student_age = 20
user_email = 'user@example.com'
print(student_age)
```

```
# Example 2: Function naming (snake_case)
def calculate_total_price(price, tax):
    return price + tax
total = calculate_total_price(100, 10)
```

```
# Example 3: Class naming (PascalCase)
class ShoppingCart:
    pass
cart = ShoppingCart()
```

```
# Example 4: Constant naming (UPPER_CASE)
MAX_CONNECTIONS = 100
DEFAULT_TIMEOUT = 30
print(MAX_CONNECTIONS)
```

```
# Example 5: Private variable (single underscore)
class Person:
    def __init__(self):
        self._age = 25
p = Person()
print(p._age)
```

```
# Example 6: Name mangling (double underscore)
class BankAccount:
    def __init__(self):
        self.__balance = 1000
account = BankAccount()
```

```
# Example 7: Magic methods (dunder)
class MyClass:
    def __init__(self):
        pass
    def __str__(self):
        return 'MyClass instance'
obj = MyClass()
```

```
# Example 8: Module naming
# filename: my_module.py
module_version = '1.0'
```

```
# Example 9: Package naming
# package name: my_package
print('Package example')
```

```
# Example 10: Boolean variable naming
is_active = True
```

```
has_permission = False
can_edit = True
print(is_active)
```

```
# Example 11: Function returning boolean
def is_valid_email(email):
    return '@' in email
print(is_valid_email('test@example.com'))
```

```
# Example 12: Descriptive names
total_number_of_students = 50
average_score = 85.5
print(total_number_of_students)
```

```
# Example 13: Avoid single letters (except loops)
for index in range(5):
    print(index)
```

```
# Example 14: Use verbs for functions
def get_user_name():
    return 'John'
def set_user_age(age):
    pass
```

```
# Example 15: Acronyms in names
HTTP_STATUS_OK = 200
JSON_DATA = {'key': 'value'}
print(HTTP_STATUS_OK)
```

6. User Input

User input allows programs to interact with users by accepting data during runtime. Python's input() function reads data from the user as a string. This data can then be processed, converted to other types, and used in the program.

6.1 Basic input() Function

The input() function displays a prompt (if provided) and waits for the user to enter data. It always returns the entered data as a string, regardless of what the user types.

```
# Example 1: Simple input
name = input('Enter your name: ')
print(f'Hello, {name}!')

# Example 2: Input without prompt
data = input()
print(f'You entered: {data}')

# Example 3: Input with message
age = input('How old are you? ')
print(f'You are {age} years old')

# Example 4: Multiple inputs
first = input('First name: ')
last = input('Last name: ')
print(f'Full name: {first} {last}')

# Example 5: Input for calculation
num = input('Enter a number: ')
print(f'You entered: {num}')
print(f'Type: {type(num)}')

# Example 6: Input with newline in prompt
response = input('Enter yes or no:\n')
print(f'Response: {response}')

# Example 7: Storing input in variable
user_input = input('Type something: ')
print(f'Length: {len(user_input)}')

# Example 8: Input with formatted string
color = input(f'What is your favorite color? ')
print(f'Your favorite color is {color}')

# Example 9: Input comparison
password = input('Enter password: ')
if password == 'secret':
    print('Access granted')

# Example 10: Input upper/lower case
text = input('Enter text: ')
print(text.upper())

# Example 11: Input with strip
name = input('Enter name: ').strip()
print(f'Trimmed name: {name}')

# Example 12: Empty input check
value = input('Enter value: ')
if value:
    print('You entered:', value)
else:
    print('No input')

# Example 13: Input in loop
for i in range(3):
    data = input(f'Enter item {i+1}: ')
    print(f'Got: {data}')

# Example 14: Input with default suggestion
choice = input('Choose (yes/no): ')
print(f'You chose: {choice}')
```

```
# Example 15: Long prompt message
feedback = input('Please enter your feedback here: ')
print(f'Thank you for: {feedback}')
```

6.2 Converting Input Types

Since input() returns a string, you often need to convert it to other types (int, float, etc.) using type conversion functions. This is essential for mathematical operations and comparisons.

```
# Example 1: Converting to integer
age = int(input('Enter age: '))
print(f'Next year you will be {age + 1}')

# Example 2: Converting to float
price = float(input('Enter price: '))
print(f'Price with tax: {price * 1.1}')

# Example 3: Converting to boolean
response = input('True or False: ')
is_true = bool(response)
print(is_true)

# Example 4: Multiple integer inputs
num1 = int(input('First number: '))
num2 = int(input('Second number: '))
print(f'Sum: {num1 + num2}')

# Example 5: Float calculation
radius = float(input('Enter radius: '))
area = 3.14 * radius ** 2
print(f'Area: {area}')

# Example 6: Input in expression
result = int(input('Enter number: ')) * 2
print(f'Double: {result}')

# Example 7: Type conversion with error
try:
    num = int(input('Enter number: '))
    print(num)
except ValueError:
    print('Invalid input')

# Example 8: Converting list input
numbers = input('Enter numbers (space-separated): ')
num_list = [int(x) for x in numbers.split()]
print(num_list)

# Example 9: Float with rounding
value = float(input('Enter decimal: '))
print(f'Rounded: {round(value, 2)}')

# Example 10: Integer division with input
total = int(input('Total items: '))
groups = int(input('Number of groups: '))
print(f'Per group: {total // groups}')

# Example 11: Converting to complex
real = float(input('Real part: '))
imag = float(input('Imaginary part: '))
complex_num = complex(real, imag)
print(complex_num)

# Example 12: String to list conversion
items = input('Enter items (comma-separated): ')
item_list = items.split(',')
print(item_list)

# Example 13: Integer with validation
while True:
    try:
        age = int(input('Enter age: '))
        break
    except:
        print('Invalid! Try again')
```

```
# Example 14: Float percentage
percent = float(input('Enter percentage: '))
decimal = percent / 100
print(f'Decimal: {decimal}')
```

```
# Example 15: Multiple type conversions
height = float(input('Height (m): '))
weight = int(input('Weight (kg): '))
bmi = weight / (height ** 2)
print(f'BMI: {bmi:.2f}')
```

6.3 Input Validation and Error Handling

It's important to validate user input to prevent errors and ensure data integrity. This includes checking for correct types, ranges, and formats using conditionals and exception handling.

```
# Example 1: Checking empty input
name = input('Enter name: ')
if name:
    print(f'Hello {name}')
else:
    print('Name cannot be empty')
```

```
# Example 2: Numeric validation
age = input('Enter age: ')
if age.isdigit():
    print(f'Valid age: {age}')
else:
    print('Please enter a number')
```

```
# Example 3: Range validation
age = int(input('Enter age: '))
if 0 <= age <= 120:
    print('Valid age')
else:
    print('Invalid age range')
```

```
# Example 4: Try-except for conversion
try:
    num = int(input('Enter number: '))
    print(f'Number: {num}')
except ValueError:
    print('Not a valid number')
```

```
# Example 5: Yes/No validation
response = input('Continue? (yes/no): ').lower()
if response in ['yes', 'no']:
    print(f'You chose: {response}')
else:
    print('Invalid response')
```

```
# Example 6: Multiple choice validation
choice = input('Choose (1/2/3): ')
if choice in ['1', '2', '3']:
    print(f'Valid choice: {choice}')
else:
    print('Invalid choice')
```

```
# Example 7: Email validation (basic)
email = input('Enter email: ')
if '@' in email and '.' in email:
    print('Email format looks valid')
else:
    print('Invalid email format')
```

```
# Example 8: Length validation
password = input('Enter password: ')
if len(password) >= 8:
    print('Password accepted')
else:
    print('Password too short')
```

```
# Example 9: Alphabetic validation
name = input('Enter name: ')
if name.isalpha():
```

```
    print('Valid name')
else:
    print('Name should contain only letters')
```

```
# Example 10: Positive number validation
try:
    num = float(input('Enter positive number: '))
    if num > 0:
        print('Valid')
    else:
        print('Must be positive')
except:
    print('Invalid input')
```

```
# Example 11: Input retry loop
while True:
    age = input('Enter age (0-120): ')
    if age.isdigit() and 0 <= int(age) <= 120:
        print('Valid!')
        break
    print('Try again')
```

```
# Example 12: Multiple validation conditions
username = input('Enter username: ')
if len(username) >= 3 and username.isalnum():
    print('Username valid')
else:
    print('Username invalid')
```

```
# Example 13: Case-insensitive validation
response = input('Enter (yes/no): ').lower().strip()
if response == 'yes':
    print('Confirmed')
elif response == 'no':
    print('Cancelled')
else:
    print('Invalid')
```

```
# Example 14: Whitespace validation
text = input('Enter text: ').strip()
if text:
    print('Input accepted')
else:
    print('Empty input')
```

```
# Example 15: Custom validation function
def validate_age(age_str):
    try:
        age = int(age_str)
        return 0 <= age <= 120
    except:
        return False

age_input = input('Age: ')
if validate_age(age_input):
    print('Valid')
else:
    print('Invalid')
```

7. Type Conversion

Type conversion (also called type casting) is the process of converting a value from one data type to another. Python supports both implicit (automatic) and explicit (manual) type conversion. This is essential when performing operations that require specific data types.

7.1 Implicit Type Conversion

Python automatically converts one data type to another when needed. This typically happens when mixing data types in operations, where Python converts to the 'wider' type to prevent data loss.

```
# Example 1: Int to float in addition
num_int = 10
num_float = 3.5
result = num_int + num_float
print(result, type(result))

# Example 2: Int to float in division
result = 10 / 2
print(result, type(result))

# Example 3: Mixed arithmetic
x = 5
y = 2.0
z = x * y
print(z, type(z))

# Example 4: Boolean to int
total = 10 + True
print(total, type(total))

# Example 5: Boolean to int (False)
total = 10 + False
print(total)

# Example 6: Int and float subtraction
result = 20 - 3.5
print(result, type(result))

# Example 7: Power operation
result = 2 ** 3.0
print(result, type(result))

# Example 8: Complex and int
complex_num = 3 + 4j
result = complex_num + 5
print(result, type(result))

# Example 9: Modulo with float
result = 10 % 3.0
print(result, type(result))

# Example 10: Mixed multiplication
result = 5 * 2.5 * True
print(result)

# Example 11: Integer division (no conversion)
result = 10 // 3
print(result, type(result))

# Example 12: Float in comparison
result = (5 > 3.0)
print(result, type(result))

# Example 13: Boolean in multiplication
result = 10 * False
print(result, type(result))

# Example 14: Implicit in complex expression
result = (10 + 5.5) * True
print(result, type(result))
```

```
# Example 15: Multiple mixed types
result = 10 + 5.5 + True + 0j
print(result, type(result))
```

7.2 Explicit Type Conversion - Numeric Types

Explicit conversion uses built-in functions like int(), float(), and complex() to manually convert between numeric types. This gives you control over type conversion.

```
# Example 1: Float to int
float_num = 9.7
int_num = int(float_num)
print(int_num, type(int_num))

# Example 2: Int to float
x = 5
float_x = float(x)
print(float_x, type(float_x))

# Example 3: String to int
num_str = '123'
num = int(num_str)
print(num, type(num))

# Example 4: String to float
price_str = '19.99'
price = float(price_str)
print(price, type(price))

# Example 5: Int to complex
num = 5
complex_num = complex(num)
print(complex_num, type(complex_num))

# Example 6: Two arguments to complex
complex_num = complex(3, 4)
print(complex_num, type(complex_num))

# Example 7: String to int with base
binary_str = '1010'
num = int(binary_str, 2)
print(num)

# Example 8: Hex string to int
hex_str = 'FF'
num = int(hex_str, 16)
print(num)

# Example 9: Float truncation
float_num = 7.9
truncated = int(float_num)
print(truncated)

# Example 10: Negative float to int
float_num = -5.8
int_num = int(float_num)
print(int_num)

# Example 11: Scientific notation string to float
sci_str = '2.5e3'
num = float(sci_str)
print(num)

# Example 12: Boolean to int
bool_val = True
int_val = int(bool_val)
print(int_val)

# Example 13: Boolean to float
bool_val = False
float_val = float(bool_val)
print(float_val)

# Example 14: Float to complex
float_num = 3.5
```

```
complex_num = complex(float_num)
print(complex_num)
```

```
# Example 15: String with spaces to float
num_str = '  42.5  '
num = float(num_str.strip())
print(num)
```

7.3 Explicit Type Conversion - Other Types

Python provides functions to convert between various types including str(), bool(), list(), tuple(), set(), and dict(). These are useful for data manipulation and formatting.

```
# Example 1: Int to string
num = 42
str_num = str(num)
print(str_num, type(str_num))
```

```
# Example 2: Float to string
price = 19.99
price_str = str(price)
print(price_str, type(price_str))
```

```
# Example 3: List to string
my_list = [1, 2, 3]
list_str = str(my_list)
print(list_str, type(list_str))
```

```
# Example 4: String to list
text = 'Python'
char_list = list(text)
print(char_list)
```

```
# Example 5: Tuple to list
my_tuple = (1, 2, 3)
my_list = list(my_tuple)
print(my_list, type(my_list))
```

```
# Example 6: List to tuple
my_list = [1, 2, 3]
my_tuple = tuple(my_list)
print(my_tuple, type(my_tuple))
```

```
# Example 7: List to set
numbers = [1, 2, 2, 3, 3, 3]
unique = set(numbers)
print(unique)
```

```
# Example 8: String to set
text = 'hello'
char_set = set(text)
print(char_set)
```

```
# Example 9: Set to list
my_set = {1, 2, 3}
my_list = list(my_set)
print(my_list)
```

```
# Example 10: Int to bool
num = 5
bool_val = bool(num)
print(bool_val, type(bool_val))
```

```
# Example 11: Zero to bool
num = 0
bool_val = bool(num)
print(bool_val)
```

```
# Example 12: Empty string to bool
empty = ''
bool_val = bool(empty)
print(bool_val)
```

```
# Example 13: Non-empty string to bool
text = 'Python'
bool_val = bool(text)
print(bool_val)
```

```
# Example 14: List of tuples to dict
pairs = [('a', 1), ('b', 2)]
my_dict = dict(pairs)
print(my_dict)
```

```
# Example 15: Range to list
range_obj = range(5)
num_list = list(range_obj)
print(num_list)
```

8. Literals

Literals are raw data values given to variables or constants in source code. They represent fixed values that don't change. Python supports various types of literals including numeric, string, boolean, special, and collection literals.

8.1 Numeric Literals

Numeric literals include integers (decimal, binary, octal, hexadecimal), floating-point numbers, and complex numbers. They can be written in different number systems and formats.

```
# Example 1: Decimal integer literal
num = 42
print(num)

# Example 2: Binary literal (prefix 0b)
binary = 0b1010
print(binary)

# Example 3: Octal literal (prefix 0o)
octal = 0o17
print(octal)

# Example 4: Hexadecimal literal (prefix 0x)
hex_num = 0xFF
print(hex_num)

# Example 5: Float literal
price = 19.99
print(price)

# Example 6: Float with exponent
sci_num = 2.5e3
print(sci_num)

# Example 7: Negative exponent
small = 1.5e-4
print(small)

# Example 8: Complex literal
complex_num = 3 + 4j
print(complex_num)

# Example 9: Complex with float
complex_num = 2.5 + 3.5j
print(complex_num)

# Example 10: Negative integer literal
num = -100
print(num)

# Example 11: Large integer literal
big = 12345678901234567890123456789O
print(big)

# Example 12: Float starting with dot
fraction = .5
print(fraction)

# Example 13: Float ending with dot
whole = 5.
print(whole)

# Example 14: Underscore in numbers (readability)
large = 1_000_000
print(large)

# Example 15: Zero literals
zero_int = 0
zero_float = 0.0
zero_complex = 0j
print(zero_int, zero_float, zero_complex)
```

8.2 String Literals

String literals are sequences of characters enclosed in single, double, or triple quotes. They can include escape sequences for special characters and support various string formats.

```
# Example 1: Single quote string
text = 'Hello World'
print(text)

# Example 2: Double quote string
message = "Python Programming"
print(message)

# Example 3: Triple quote string
long_text = '''This is a
multi-line
string'''
print(long_text)

# Example 4: Triple double quotes
docstring = """This is also
a multi-line string"""
print(docstring)

# Example 5: Escape sequence - newline
text = 'Hello\nWorld'
print(text)

# Example 6: Escape sequence - tab
text = 'Name:\tJohn'
print(text)

# Example 7: Escape sequence - backslash
path = 'C:\\Users\\Documents'
print(path)

# Example 8: Escape sequence - quotes
text = 'He said, \'Hello\''
print(text)

# Example 9: Raw string literal
path = r'C:\new\folder'
print(path)

# Example 10: Unicode escape
text = '\u2665 Heart'
print(text)

# Example 11: Hex escape
text = '\x48\x65\x6C\x6C\x6F'
print(text)

# Example 12: Empty string
empty = ''
print(repr(empty))

# Example 13: String with quotes
text = "He said, 'Python is great!'"
print(text)

# Example 14: Byte string literal
byte_str = b'Hello'
print(byte_str, type(byte_str))

# Example 15: F-string literal
name = 'Alice'
text = f'Hello, {name}!'
print(text)
```

8.3 Boolean and Special Literals

Boolean literals (True and False) represent truth values. The special literal None represents the absence of a value. These are fundamental to control flow and data representation.

```
# Example 1: True literal
is_active = True
print(is_active, type(is_active))

# Example 2: False literal
is_finished = False
print(is_finished)

# Example 3: None literal
value = None
print(value, type(value))

# Example 4: Boolean in condition
if True:
    print('Always executes')

# Example 5: Boolean assignment
flag = False
flag = not flag
print(flag)

# Example 6: None as default
def my_function(param=None):
    if param is None:
        print('No parameter')
my_function()

# Example 7: Checking for None
result = None
if result is None:
    print('Result is None')

# Example 8: Boolean from comparison
is_greater = (10 > 5)
print(is_greater)

# Example 9: Multiple None assignments
x = y = z = None
print(x, y, z)

# Example 10: Boolean in list
flags = [True, False, True]
print(flags)

# Example 11: None in dictionary
data = {'key': None}
print(data)

# Example 12: Boolean operations
result = True and False
print(result)

# Example 13: Boolean or operation
result = False or True
print(result)

# Example 14: Not operation
result = not False
print(result)

# Example 15: Identity comparison
if None is None:
    print('Identity check passes')
```

8.4 Collection Literals

Collection literals define built-in data structures like lists, tuples, sets, and dictionaries directly in code. They provide a concise way to create and initialize collections.

```
# Example 1: List literal
fruits = ['apple', 'banana', 'cherry']
print(fruits)

# Example 2: Empty list literal
empty_list = []
```

```
print(empty_list)

# Example 3: Tuple literal
coordinates = (10, 20)
print(coordinates, type(coordinates))

# Example 4: Empty tuple literal
empty_tuple = ()
print(empty_tuple)

# Example 5: Single element tuple
single = (5,)
print(single, type(single))

# Example 6: Set literal
unique_nums = {1, 2, 3, 3, 4}
print(unique_nums)

# Example 7: Empty set (not {})
empty_set = set()
print(empty_set, type(empty_set))

# Example 8: Dictionary literal
person = {'name': 'John', 'age': 25}
print(person)

# Example 9: Empty dictionary literal
empty_dict = {}
print(empty_dict, type(empty_dict))

# Example 10: Nested list literal
matrix = [[1, 2], [3, 4], [5, 6]]
print(matrix)

# Example 11: List with mixed types
mixed = [1, 'hello', 3.14, True]
print(mixed)

# Example 12: Dictionary with various types
data = {'int': 42, 'str': 'text', 'list': [1, 2]}
print(data)

# Example 13: Tuple unpacking
x, y = (10, 20)
print(x, y)

# Example 14: Set with strings
colors = {'red', 'green', 'blue'}
print(colors)

# Example 15: Dictionary comprehension literal
squares = {x: x**2 for x in range(5)}
print(squares)
```