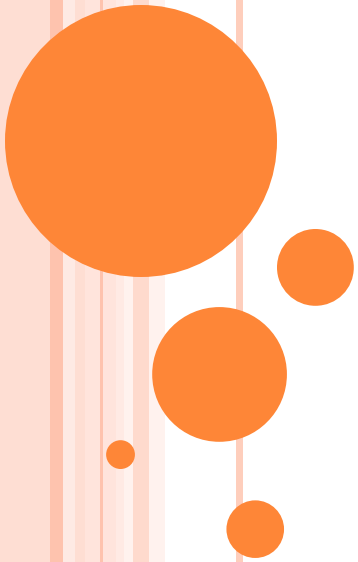
Introduction to Python Programming



CONTENTS

1.2 Input/output

Learning objectives

By the end of this section you should be able to

- Display output using the `print()` function.
- Obtain user input using the `input()` function.

Basic output

The **`print()`** function displays output to the user. **Output** is the information or result produced by a program.
The sep and end options can be used to customize the output. Table 1.1 shows examples of sep and end.

Code	Output
`print("Today is Monday.")` `print("I like string beans.")`	`Today is Monday.` `I like string beans.`
`print("Today", "is", "Monday")` `print("Today", "is", "Monday", sep="...")`	`Today is Monday` `Today...is...Monday`
`print("Today is Monday, ", end="")` `print("I like string beans.")`	`Today is Monday, I like string beans.`
`print("Today", "is", "Monday", sep="? ", end="!!")` `print("I like string beans.")`	`Today? is? Monday!!I like string beans.`

CONCEPTS IN PRACTICE

The print() function

1. Which line of code prints `Hello world!` as one line of output?
   a. `print(Hello world!)`
   b. `print("Hello", "world", "!")`
   c. `print("Hello world!")`

2. Which lines of code prints `Hello world!` as one line of output?
   a.
   ```
   print("Hello")
   print(" world!")
   ```
   b.
   ```
   print("Hello")
   print(" world!", end="")
   ```
   c.
   ```
   print("Hello", end="")
   print(" world!")
   ```

3. What output is produced by the following statement?

   ```
   print("555", "0123", sep="-")
   ```

   a. `555 0123`
   b. `5550123-`
   c. `555-0123`

Basic input

Computer programs often receive input from the user. **Input** is what a user enters into a program. An input statement, `variable = input("prompt")`, has three parts:

1. A **variable** refers to a value stored in memory. In the statement above, *variable* can be replaced with any name the programmer chooses.
2. The **input()** function reads one line of input from the user. A function is a named, reusable block of code that performs a task when called. The input is stored in the computer's memory and can be accessed later using the variable.
3. A **prompt** is a short message that indicates the program is waiting for input. In the statement above, *"prompt"* can be omitted or replaced with any message.

CONCEPTS IN PRACTICE

The input() function

4. Which line of code correctly obtains and stores user input?
   a. `input()`
   b. `today_is = input`
   c. `today_is = input()`

5. Someone named Sophia enters their name when prompted with

```
print("Please enter your name: ")
name = input()
```

What is displayed by `print("You entered:", name)`?
   a. `You entered: name`
   b. `You entered: Sophia`

   c. `You entered:, Sophia`

6. What is the output if the user enters "six" as the input?

```
print("Please enter a number: ")
number = input()
print("Value =", number)
```

   a. `Value = six`
   b. `Value = 6`
   c. `Value = number`

1.3 Variables

Learning objectives

By the end of this section you should be able to

- Assign variables and print variables.
- Explain rules for naming variables.

Assignment statement

Variables allow programs to refer to values using names rather than memory locations. Ex: age refers to a person's age, and `birth` refers to a person's date of birth.

A statement can set a variable to a value using the **assignment operator** (=). Note that this is different from the equal sign of mathematics. Ex: `age = 6` or `birth = "May 15"`. The left side of the assignment statement is a variable, and the right side is the value the variable is assigned.

CONCEPTS IN PRACTICE

Assigning and using variables

1. Which line of code correctly retrieves the value of the variable, `city`, after the following assignment?

```
city = "Chicago"
```

   a. `print("In which city do you live?")`
   b. `city = "London"`
   c. `print("The city where you live is", city)`

2. Which program stores and retrieves a variable correctly?

   a.
```
print("Total =", total)
total = 6
```
   b.
```
total = 6
print("Total =", total)
```
   c.
```
print("Total =", total)

total = input()
```

3. Which is the assignment operator?

   a. `:`
   b. `==`
   c. `=`

4. Which is a valid assignment?

   a. `temperature = 98.5`
   b. `98.5 = temperature`
   c. `temperature - 23.2`

Variable naming rules

A variable name can consist of letters, digits, and underscores and be of any length. The name cannot start with a digit. Ex: `101class` is invalid. Also, letter case matters. Ex: `Total` is different from `total`. Python's style guide recommends writing variable names in **snake case**, which is all lowercase with underscores in between each word, such as `first_name` or `total_price`.

A name should be short and descriptive, so words are preferred over single characters in programs for readability. Ex: A variable named `count` indicates the variable's purpose better than a variable named `c`.

Python has reserved words, known as **keywords**, which have special functions and cannot be used as names for variables (or other objects).

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

CONCEPTS IN PRACTICE

Valid variable names

5. Which can be used as a variable name?
   a. median
   b. class
   c. import

6. Why is the name, `2nd_input`, not a valid variable name?
   a. contains an underscore
   b. starts with a digit
   c. is a keyword

7. Which would be a good name for a variable storing a zip code?
   a. `z`
   b. `var_2`
   c. `zip_code`

8. Given the variable name, `DogBreed`, which improvement conforms to Python's style guide?
   a. `dog_breed`
   b. `dogBreed`
   c. `dog-breed`

1.4 String basics

Learning objectives

By the end of this section you should be able to

- Use the built-in `len()` function to get a string's length.
- Concatenate string literals and variables using the + operator.

Quote marks

A string is a sequence of characters enclosed by matching single (') or double (") quotes. Ex: `"Happy birthday!"` and `'21'` are both strings.

To include a single quote (') in a string, enclose the string with matching double quotes ("). Ex: `"Won't this work?"` To include a double quote ("), enclose the string with matching single quotes ('). Ex: `'They said "Try it!", so I did'`.

Valid string	Invalid string
`"17" or '17'`	`17`
`"seventeen" or 'seventeen'`	`seventeen`
`"Where?" or 'Where?'`	`"Where?'`
`"I hope you aren't sad."`	`'I hope you aren't sad.'`
`'The teacher said "Correct!" '`	`"The teacher said "Correct!" "`

CONCEPTS IN PRACTICE

Valid and invalid strings

1. Which of the following is a string?
   a. `Hello!`
   b. `29`
   c. `"7 days"`

2. Which line of code assigns a string to the variable `email`?
   a. `"fred78@gmail.com"`
   b. `"email = fred78@gmail.com"`
   c. `email = "fred78@gmail.com"`

3. Which is a valid string?
   a. `I know you'll answer correctly!`
   b. `'I know you'll answer correctly!'`
   c. `"I know you'll answer correctly!"`

4. Which is a valid string?
   a. `You say "Please" to be polite`
   b. `"You say "Please" to be polite"`
   c. `'You say "Please" to be polite'`

len() function

A common operation on a string object is to get the string length, or the number of characters in the string. The **len()** function, when called on a string value, returns the string length.

CONCEPTS IN PRACTICE

Applying len() function to string values

5. What is the return value for `len("Hi Ali")`?
   a. `2`
   b. `5`
   c. `6`

6. What is the length of an empty string variable (`""`)?
   a. undefined
   b. 0
   c. 2

7. What is the output of the following code?

```
number = "12"
number_of_digits = len(number)
print("Number", number, "has", number_of_digits, "digits.")
```

   a. `Number 12 has 12 digits.`
   b. `Number 12 has 2 digits.`
   c. `Number 12 has number_of_digits digits.`

Concatenation

Concatenation is an operation that combines two or more strings sequentially with the concatenation operator (+). Ex: `"A" + "part"` produces the string `"Apart"`.

CONCEPTS IN PRACTICE

String concatenation

8. Which produces the string `"10"`?

a. `1 + 0`
b. `"1 + 0"`
c. `"1" + "0"`

9. Which produces the string `"Awake"`?

a. `"wake" + "A"`
b. `"A + wake"`
c. `"A" + "wake"`

10. A user enters `"red"` after the following line of code.

```
color = input("What is your favorite color?")
```

Which produces the output `"Your favorite color is red!"`?

a. `print("Your favorite color is " + color + !)`
b. `print("Your favorite color is " + "color" + "!")`
c. `print("Your favorite color is " + color + "!")`

11. Which of the following assigns `"one-sided"` to the variable `holiday`?

a. `holiday = "one" + "sided"`
b. `holiday = one-sided`
c. `holiday = "one-" + "sided"`

1.5 Number basics

Learning objectives

By the end of this section you should be able to

- Use arithmetic operators to perform calculations.
- Explain the precedence of arithmetic operators.

Numeric data types

Python supports two basic number formats, integer and floating-point. An integer represents a whole number, and a floating-point format represents a decimal number. The format a language uses to represent data is called a **data type**. In addition to integer and floating-point types, programming languages typically have a string type for representing text.

CONCEPTS IN PRACTICE

Integers, floats, and strings

Assume that `x = 1`, `y = 2.0`, and `s = "32"`.

1. What is the output of the following code?

```
print(x, type(x))
```

   a. `1 'int'.`
   b. `1.0 <class 'float'>.`
   c. `1 <class 'int'>.`

2. What is the output of the following code?

```
print(y, type(y))
```

   a. `2.0 <class 'int'>`
   b. `2.0 <class 'float'>`
   c. `2 <class 'int'>`

3. What is the type of the following value?

```
"12.0"
```

   a. `string`
   b. `int`
   c. `float`

Basic arithmetic

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication, and division.

Four basic arithmetic operators exist in Python:

1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)

CONCEPTS IN PRACTICE

Applying arithmetic operators

Assume that `x = 7`, `y = 20`, and `z = 2`.

4. Given the following lines of code, what is the output of the code?

```
c = 0
c = x - z
c = c + 1
print(c)
```

a. `1`
b. `5`
c. `6`

5. What is the value of a?

```
a = 3.5 - 1.5
```

a. `2`
b. `2.0`
c. `2.5`

6. What is the output of `print(x / z)`?
   a. `3`
   b. `3.0`
   c. `3.5`

7. What is the output of `print(y / z)`?
   a. `0`
   b. `10`
   c. `10.0`

8. What is the output of `print(z * 1.5)`?
   a. `2`
   b. `3`
   c. `3.0`

Operator precedence

When a calculation has multiple operators, each operator is evaluated in order of **precedence**. Ex: `1 + 2 * 3` is `7` because multiplication takes precedence over addition. However, `(1 + 2) * 3` is `9` because parentheses take precedence over multiplication.

Operator	Description	Example	Result
`()`	Parentheses	`(1 + 2) * 3`	`9`
`**`	Exponentiation	`2 ** 4`	`16`
`+, -`	Positive, negative	`-math.pi`	`-3.14159`
`*, /`	Multiplication, division	`2 * 3`	`6`
`+, -`	Addition, subtraction	`1 + 2`	`3`

CONCEPTS IN PRACTICE

Multiple arithmetic operators

9. What is the value of `4 * 3 ** 2 + 1`?

a. `37`
b. `40`
c. `145`

10. Which part of `(1 + 3) ** 2 / 4` evaluates first?

a. `4 ** 2`
b. `1 + 3`
c. `2 / 4`

11. What is the value of `-4 ** 2`?

a. `-16`
b. `16`

12. How many operators are in the following statement?

```
result = -2 ** 3
```

a. 1
b. 2
c. 3

1.6 Error messages

Learning objectives

By the end of this section you should be able to

- Identify the error type and line number in error messages.
- Correct syntax errors, name errors, and indentation errors.

How to read errors

A natural part of programming is making mistakes. Even experienced programmers make mistakes when writing code. Errors may result when mistakes are made when writing code. The computer requires very specific instructions telling the computer what to do. If the instructions are not clear, then the computer does not know what to do and gives back an error.

When an error occurs, Python displays a message with the following information:

1. The line number of the error.
2. The type of error (Ex: `SyntaxError`).
3. Additional details about the error.

Ex: Typing `print "Hello!"` without parentheses is a syntax error. In Python, parentheses are required to use `print`. When attempting to run `print "Hello!"`, Python displays the following error:

```
Traceback (most recent call last):
  File "/home/student/Desktop/example.py", line 1
    print "Hello"
                ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print("Hello")?
```

CONCEPTS IN PRACTICE

Parts of an error

Given the following error message:

```
Traceback (most recent call last):
  File "/home/student/Desktop/example.py", line 2
    print "test"
        ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print("test")?
```

1. What is the filename of the program?
   a. Desktop
   b. example.py
   c. test

2. On which line was the error found?
   a. 1
   b. 2
   c. 3

3. What type of error was found?
   a. missing parentheses
   b. `SyntaxError`
   c. traceback

Common types of errors

Different types of errors may occur when running Python programs. When an error occurs, knowing the type of error gives insight about how to correct the error. The following table shows examples of mistakes that anyone could make when programming.

Mistake	Error message	Explanation
`print("Have a nice day!"`	`SyntaxError: unexpected EOF while parsing`	The closing parenthesis is missing. Python is surprised to reach the end of file (EOF) before this line is complete.
`word = input("Type a word: )`	`SyntaxError: EOL while scanning string literal`	The closing quote marks are missing. As a result, the string does not terminate before the end of line (EOL).
`print("You typed:", wird)`	`NameError: name 'wird' is not defined`	The spelling of `word` is incorrect. The programmer accidentally typed the wrong key.

Mistake	Error message	Explanation
`prints("You typed:", word)`	`NameError: name 'prints' is not defined`	The spelling of `print` is incorrect. The programmer accidentally typed an extra letter.
`  print("Hello")`	`IndentationError: unexpected indent`	The programmer accidentally typed a space at the start of the line.
`    print("Goodbye")`	`IndentationError: unexpected indent`	The programmer accidentally pressed the Tab key at the start of the line.

CONCEPTS IN PRACTICE

Types of errors

For each program below, what type of error will occur?

4.
```
print("Breakfast options:")
 print("A. Cereal")
 print("B. Eggs")
 print("C. Yogurt")
choice = input("What would you like? ")
```

a. IndentationError
b. NameError
c. SyntaxError

5.
```
birth = input("Enter your birth date: )
print("Happy birthday on ", birth)
```

a. IndentationError
b. NameError
c. SyntaxError

6.
```
print("Breakfast options:")
print(" A. Cereal")
print(" B. Eggs")
print(" C. Yogurt")
choice = intput("What would you like? ")
```

a. IndentationError
b. NameError
c. SyntaxError

1.7 Comments

Learning objectives

By the end of this section you should be able to

- Write concise, meaningful comments that explain intended functionality of the code.
- Write a docstring (more verbose comment) that describes the program functionality.

The hash character

Comments are short phrases that explain what the code is doing. Ex: Lines 1, 8, and 10 in the following program contain comments. Each comment begins with a hash character (`#`). All text from the hash character to the end of the line is ignored when running the program. In contrast, hash characters inside of strings are treated as regular text. Ex: The string `"Item #1: "` does not contain a comment.

When writing comments:

- The `#` character should be followed by a single space. Ex: `# End of menu` is easier to read than `#End of menu`.
- Comments should explain the purpose of the code, not just repeat the code itself. Ex: `# Get the user's preferences` is more descriptive than `# Input item1 and item2`.

EXAMPLE 1.2

Program with three comments

```
 1 # Display the menu options
 2 print("Lunch Menu")
 3 print("----------")
 4 print("Burrito")
 5 print("Enchilada")
 6 print("Taco")
 7 print("Salad")
 8 print() # End of menu
 9
10 # Get the user's preferences
11 item1 = input("Item #1: ")
12 item2 = input("Item #2: ")
```

CONCEPTS IN PRACTICE

Simple comments

1. The main purpose of writing comments is to _____.
   a. avoid writing syntax errors
   b. explain what the code does
   c. make the code run faster

2. Which symbol is used for comments in Python?
   a. #
   b. /*
   c. //

3. Which comment is formatted correctly?
   a. 0 spaces:
   ```
   #Get the user input
   ```
   b. 1 space:
   ```
   # Get the user input
   ```
   c. 2 spaces:
   ```
   #  Get the user input
   ```

1.9 Chapter summary

This chapter introduced the basics of programming in Python, including:

- `print()` and `input()`.
- Variables and assignment.
- Strings, integers, and floats.
- Arithmetic, concatenation.
- Common error messages.
- Comments and docstrings.

At this point, you should be able to write programs that ask for input, perform simple calculations, and output the results. The programming practice below ties together most topics presented in the chapter.

Function	Description
`print(values)`	Outputs one or more values, each separated by a space, to the user.
`input(prompt)`	If present, `prompt` is output to the user. The function then reads a line of input from the user.
`len(string)`	Returns the length (the number of characters) of a string.
`type(value)`	Returns the type (or class) of a value. Ex: `type(123)` is `<class 'int'>`.
Operator	**Description**
= (Assignment)	Assigns (or updates) the value of a variable. In Python, variables begin to exist when assigned for the first time.

Function	Description
+ (Concatenation)	Appends the contents of two strings, resulting in a new string.
+ (Addition)	Adds the values of two numbers.
- (Subtraction)	Subtracts the value of one number from another.
* (Multiplication)	Multiplies the values of two numbers.
/ (Division)	Divides the value of one number by another.
** (Exponentiation)	Raises a number to a power. Ex: `3**2` is three squared.
Syntax	**Description**
# (Comment)	All text is ignored from the # symbol to the end of the line.
' or " (String)	Strings may be written using either kind of quote. Ex: `'A'` and `"A"` represent the same string. By convention, this book uses double quotes (") for most strings.
""" (Docstring)	Used for documentation, often in multi-line strings, to summarize a program's purpose or usage.