CSI-160 Python Programming
Python Fundamentals

Vikas Thammanna Gowda

07/10/2025

Contents

1 Variable

A *variable* in Python is a name that refers to a value stored in memory. Think of it as a labeled container that holds data. Python is dynamically typed, so you don't declare the type explicitly—variables can be rebound to objects of different types at runtime.

Variable Creation and Assignment

```
value_x = 10     # value_x refers to an integer object with value 10
name = "Alice"   # name refers to a string object "Alice"
pi = 3.14159     # pi refers to a float object
```

1.1 Identifier

An *identifier* is the name you choose for variables, functions, classes, modules, etc. It must follow Python's naming conventions and syntax rules.

1.2 Rules for Identifiers

Python enforces several rules for valid identifiers:

1. **Start with a letter or underscore**:
   - Valid: `_temp`, `varName`, `tax_1`.
   - Invalid: `1_tax` (starts with digit).
2. **Contain only letters, digits, or underscores. No spaces or special characters**:
   - Valid: `user1`, `_data_set2`.
   - Invalid: `first name`, `user-name` , `count$`.
3. **Case-sensitive**: `Data`, `DATA`, and `data` are distinct.
4. **Cannot be a Python keyword** (Inbuilt identifiers): Reserved words such as `if`, `for`, `while`, `import`. Invalid: `class = 5`.
5. **Unlimited length (but keep names reasonable)**.

2 Data types

A data type is simply a way of telling Python (and you!) what kind of information you're working with, so that the computer knows:

1. How much space to set aside in memory?
2. What operations make sense? (e.g. you can add two numbers but you can't add two pieces of text instead you can concatenate them)

Analogy: sorting school supplies
Imagine you've got bins labeled "Pen", "Erasers", "Rulers", and "Notebooks". Before you drop an item into a bin, you look at what it is:

- If it's a pen, it goes in the Pens bin.
- If you tried to put a notebook into the Erasers bin, someone would stop you!

In programming, data types are like those labeled bins. They keep your information organized so that Python knows:

- You can add two pens to get more pens.
- You can't mix a ruler with an eraser in the same operation.

2.1 Integer (`int`)

Definition: Whole numbers without any fractional part.
Analogy: Like counting the number of apples in a basket—you can't have 3.5 apples (unless you cut one!).

Assigning integers to variables

```
num_students = 25
year = 2025
```

2.2 Floating-Point Number (`float`)

Definition: Numbers that have a decimal point (can represent fractions).
Analogy: Like measuring 3.75 liters of juice—you often need decimals in real life.

Assigning floats to variables

```
price_per_hour = 12.50
temperature = 98.6
```

2.3 String (`str`)

Definition: A sequence of characters (letters, numbers, symbols) **enclosed in quotes**.
Analogy: Like writing words or sentences on a chalkboard—anything you can type.

Assigning strings to variables

```
first_name = "Alex"
greeting = 'Hello, world!'
```

2.4 Boolean (`bool`)

Definition: A value that is either `True` or `False`.
Analogy: A light switch—it's either on (`True`) or off (`False`).

Assigning boolean to variables

```
is_raining = False
passed_test = True
```

Summary

Data Type	Example Value	Use Case	Size (bytes)
`int`	42	Counting items, indexing	28
`float`	3.14	Measurements, prices	24
`str`	`"Hello"`	Text, messages	52
`bool`	`True`	Yes/no flags, conditional checks	28

Table 1: Basic data types

3 Input and Output (I/O)

When you write a Python program, it doesn't do much if it just sits there silently! Most useful programs need to communicate with the person using them:

- Input is how your program receives information from the user (for example, asking for their name or a number).
- Output is how your program shares information back (for example, printing a greeting or the result of a calculation).

Together, input and output (often called **I/O**) let you build interactive programs—ones that can ask questions, take in answers, do something with those answers, and then tell the user what happened.

3.1 The `print()` Function

Purpose: Display text (or other values) on the screen.

Syntax:

```
print(value1, value2, ..., sep = ' ', end = '\n')
```

where,

`value1, value2, ...` are the things you want to display.

`sep` is the string placed between values (default is a space).

`end` is what's printed after all values (default is a newline).

Note: **`print`** is a built-in function. **`sep`** and **`end`** are keywords.

Example 1:

```
print(10, 20, 30, sep =' ', end ='\n')
print(10, 20, 30)
print(10, 20, 30, sep =',')
print(10, 20, 30, sep ='')  # no space between the values
print("Hello, world!")
print("Sum of 2 and 3 is", 2 + 3)
```

Output:

```
10 20 30
10 20 30
10,20,30
102030
Hello, world!
Sum of 2 and 3 is 5
```

Example 2:

```
print("Zoro!", end = '\n')
print(10, 20, 30,  end =' ') # line ends with a space
print(11, 22, 33)
```

Output:

```
Zoro
10 20 30 11 22 33
```

3.2 The `input()` Function

Purpose: Pause the program and ask the user to type something; then return what they typed as a **`string`**.

Syntax:

```
user_text = input(prompt)
```

where,

`prompt` is the message shown to the user enclosed in quotes.

`user_text` stores whatever the user types (until they press Enter) as a string.

Example 1:

```
name = input("What is your name? ")
print("Hello, " + name + "!")
```

Output:

```
What is your name? Alice
Hello, Alice!
```

Step-by-step:

1. Python displays `What is your name?` and waits.

   Output:

   ```
   What is your name?
   ```

2. You type, for example, `Alice` and press Enter.

   Output:

   ```
   What is your name? Alice
   ```

3. The variable `name` now holds the string `"Alice"`.

4. `print("Hello, " + name + "!")` prints `Hello, Alice!`.
   Note: `+` is used to concatenate strings.

4 Type Casting

In programming, type casting (also called type conversion) is the process of changing a value from one data type to another. For example, you might want to turn the text `"42"` into the number `42` so you can do arithmetic with it. Type casting helps make your programs more flexible and prevents errors when you mix different kinds of data.

4.1 Why Do We Need Type Casting?

Imagine you ask the user for their age, even if they type `16`, Python stores it as a string (`"16"`).

Example 1:

```
age = input("How old are you? ")
print(type(age))
```

Output:

```
How old are you? 16
<class 'str'>
```

You can't perform arithmetic operations unless you first convert into the integer/float. That's where type casting comes in.

Example 2:

```
text = input("How old are you? ")
age = int(text)
print(type(age))
print("Next year you will be", age + 1)
```

Output:

```
How old are you? 16
<class 'int'>
Next year you will be 17
```

4.2 Common Casting Functions in Python

Function	Converts To	Example Usage
`int()`	Integer	`int("42")` → 42; `int(42.6)` → 42
`float()`	Float	`float("3.14")` → 3.14; `float(5)` → 5.0
`str()`	String	`str(99)` → "99"
`bool()`	Boolean	`bool(0)` → False; `bool("hi")` → True

Table 2: Casting Functions