Sharif University of Technology
Department of Computer Engineering

Fundamentals of Programming

Python Language

Arman Malekzadeh
PhD Candidate in Artificial Intelligence

Table of contents

Variables and Data Types

Variables

Definition

A variable is a container for storing data values.

The name of a variable in Python

- should start with a letter or the underscore character
- can only consist of letters, numbers, and underscores
- is case-sensitive
- should not be the same as a Python keyword

Note: We do not have constants in Python. However, if you want to tell others not to change the value of a variable, write its name with uppercase letters.

Variables

Accepted Variable Names:

- `my_variable`
- `variable1`
- `_hidden_variable`

Not Accepted Variable Names:

- `1variable` (Cannot start with a number)
- `-my-variable` (Hyphen is not allowed)
- `my variable` (Spaces are not allowed)

Most Common Data Types

- **Integer:** Integers are positive or negative whole numbers with no decimal point. For example, 10, -10.
- **Float:** Floats represent real numbers and are written with a decimal point dividing the integer and fractional parts. For example, 1.0, -35.59.
- **String:** Strings in Python are arrays of bytes representing Unicode characters. For example, "Hello", "Python".
- **Boolean:** Data type with two built-in values, True and False. They are used to represent truth values (other values can also be considered false or true).

Most Common Data Types

- **List:** A list is a collection which is ordered and changeable. It allows duplicate members. For example, [1,2,3], ["apple", "banana", "cherry"].
- **Tuples:** A tuple is a collection which is ordered and unchangeable (immutable). Tuples allow duplicate members. For example (1,2,3), ("apple", "banana", "cherry").
- **Dictionary:** A dictionary is a collection which is unordered, changeable and indexed. No duplicate members. For example, {"name": "John", "age": 30}.

Casting

Casting in Python is done using constructor functions:

- **int():** constructs an integer number from an integer literal, a float literal (by rounding down to the previous whole number), or a string literal (providing the string represents a whole number)
- **float():** constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
- **str():** constructs a string from a wide variety of data types, including strings, integer literals and float literals
- **bool():** constructs a boolean from a wide variety of data types, including strings, integer literals and float literals

Casting: Examples

- `x = int(1)` # x will be 1
- `y = int(2.8)` # y will be 2
- `z = int("3")` # z will be 3
- `x = float(1)` # x will be 1.0
- `y = float(2.8)` # y will be 2.8
- `z = float("3")` # z will be 3.0
- `w = float("4.2")` # w will be 4.2
- `x = str("s1")` # x will be 's1'
- `y = str(2)` # y will be '2'
- `z = str(3.0)` # z will be '3.0'

Operators

Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
**	Exponentiation
//	Floor division

Arithmetic Operators: Examples

- `x = 5` # x will be 5
- `y = 3` # y will be 3
- `print(x + y)` # Prints 8
- `print(x - y)` # Prints 2
- `print(x * y)` # Prints 15
- `print(x / y)` # Prints 1.6666666666666667
- `print(x % y)` # Prints 2
- `print(x ** y)` # Prints 125
- `print(x // y)` # Prints 1

Comparison Operators

Operator	Description
==	Equal
!=	Not equal
$>$	Greater than
$<$	Less than
$>=$	Greater than or equal to
$<=$	Less than or equal to

Comparison Operators: Examples

- `x = 5` # x will be 5
- `y = 3` # y will be 3
- `print(x == y)` # Prints False
- `print(x != y)` # Prints True
- `print(x > y)` # Prints True
- `print(x < y)` # Prints False
- `print(x >= y)` # Prints True
- `print(x <= y)` # Prints False

Logical Operators

Operator	Description
and	Returns True if both statements are true
or	Returns True if one of the statements is true
not	Reverse the result, returns False if the result is true

Logical Operators: Examples

- `x = 5` # x will be 5
- `print(x > 3 and x < 10)` # Prints True
- `print(x > 3 or x < 4)` # Prints True
- `print(not(x > 3 and x < 10))` # Prints False

Assignment Operators: Examples

- `x = 5` # x will be 5
- `x += 3` # x will be 8
- `x -= 3` # x will be 5
- `x *= 3` # x will be 15
- `x /= 3` # x will be 5.0
- `x %= 3` # x will be 2.0
- `x **= 3` # x will be 8.0
- `x //= 3` # x will be 2.0

Identity operators: Examples

- `x = ["apple", "banana"]` # x will be ["apple", "banana"]
- `y = ["apple", "banana"]` # y will be ["apple", "banana"]
- `z = x` # z will be ["apple", "banana"]
- `print(x is z)` # Prints True
- `print(x is y)` # Prints False
- `print(x == y)` # Prints True

Membership Operators: Examples

- `x = ["apple", "banana"]` # x will be ["apple", "banana"]
- `print("banana" in x)` # Prints True
- `print("pineapple" not in x)` # Prints True

Precedence of Operators: Examples

1. **Parentheses** - They have the highest precedence and can be used to force an expression to evaluate in the order you want. For example, in the expression `(2+3) * 4`, the addition operation is performed first because of the parentheses.

2. **Exponentiation** - This operator has the next highest precedence. For instance, in the expression `2**1+1`, exponentiation is performed before addition, so the result is 3.

3. **Multiplication and Division** - These operators have equal precedence and are evaluated from left to right. For example, in `2*3/2`, multiplication is done first resulting in 3.

Precedence of Operators: Examples

4. **Addition and Subtraction** - These operators also have equal precedence and are evaluated from left to right. In `5+5-2`, addition happens first, resulting in 8.

5. **Less than and Greater than** - These comparison operators have lower precedence than arithmetic operators. For example, with `(1+1) > 2-1`, arithmetic operations are performed first followed by comparison.

6. **Logical Operators** - Logical AND (`&`), OR (`|`), NOT (`~`) are evaluated after comparison operators. For instance, in `(5 > 4) | (3 == 3)`, comparisons are made first then logical OR operation occurs.

Input

Getting input from user: Examples

- `x = input("Enter your name: ")` # x will be the name entered by the user
- `x = int(input("Enter your age: "))` # x will be the age entered by the user

Conditional Statements

Conditional Statements: `if`

In Python, `if` is a conditional statement that executes some specified code after checking if its expression is `True`.

```
if expression:
    statement(s) # code to execute if condition is True
```

```
x = 10
if x > 5:
    print("x is greater than 5")
```

In this example, the condition checks whether `x` is greater than 5. If it is, it prints `x is greater than 5`.

Conditional Statements: `if`

An `if` statement can be combined with `elif` (short for `else if`) and `else` for more complex conditional checks.

```
x = 10
if x > 10:
    print("x is greater than 10")
elif x == 10:
    print("x equals to 10")
else:
    print("x is less than 10")
```

In this case, Python first checks if `x` is greater than 10. If that's not true, it moves on to the next condition (`elif x == 10`). If that's also not true, it executes the code in the `else` block.

Code

All the code for this session is available Here:

References

References I

[1] B Downey, A. (2015). Think Python: How to Think Like a Computer Scientist-2nd Edition.

[2] Deitel, H. M., & Deitel, P. J. (2004). C: How to program. Pearson Educacion.

Sharif University of Technology
Department of Computer Engineering

Arman Malekzadeh

Fundamentals of Programming

Python Language