1. Basics

October 17, 2022

1 Hello World

[ ]:
```
print("Hello World")
```

[ ]:
```
print("Hello World")
print("Good Morning")
```

2 Variables and Dynamic Typing

- A variable is a named location used to store data in the memory. It is helpful to think of variables as a container that holds data that can be changed later in the program.
- Assignment operator = is used to assign a value to a variable.
- In Python no need to declare variable and its type, like we do in C or Java
- Datatype checks (making sure variables have the correct type for an operation) is performed at runtime.
- We can obtain the type of an object with `type()` function

[ ]:
```
a = 5
```

[ ]:
```
b = "HELLO"
```

[ ]:
```
type(a)
```

[ ]:
```
type(b)
```

3 Constants

- A constant is a type of variable whose value cannot be changed.
- Constants are written in all capital letters and underscores separating the words.
- In Python, constants are usually declared and assigned in a module. Here, the module is a new file containing variables, functions, etc which is imported to the main file.

[ ]:
```
OUTPUT_PATH = "/home/user1/output_dir"
PI = 3.14
```

4 Rules and Naming Convention for Variables and Constants

- Variable names should have a combination of letters in lowercase (`a to z`) or uppercase (`A to Z`) or digits (`0 to 9`) or an underscore (_).
- For example:
  - `snake_case`
  - `camelCase`
  - `CapWords`
- Create a name that makes sense. For example, `vowel` makes more sense than `v`
- If a variable name having two words, use underscore to separate them. e.g. `my_name`
- Use capital letters to declare a constant.
  - `MACRO_CASE`
- Don't start a variable name with a digit.
- Never use special symbols like !, @, #, $, %, etc.

5 Basic Input: Input from Keyboard

Using `input()` function

[ ]:
```
a = input("Enter Name of your City: ")
```

[ ]:
```
print(a)
```

[ ]:
```
a
```

6 Comments

- To add Comment at `#` at the beginning of the comment line

[ ]:
```
# This is a comment
print("How you doin?")
```

- Inline Comments

[ ]:
```
a = 28
print(a) # This is inline comment
```

[ ]:
```
# Python has no special syntax for multiline comments.
# To use multiline comment just begin each line with #.
# Keyboard shortcut in Jupyter to comment or uncomment is Ctrl + /
# These four lines are an example of multiline comments.
```

7 Data types

Basic Datatypes 1. Integer 2. Float 3. Boolean 4. None

Compound Datatypes 1. String 2. List 3. Tuple 4. Dictionary

7.1 Integer

[ ]:
```
a = 10
```

[ ]:
```
type(a)
```

7.2 Float

[ ]:
```
b = 3.14
```

[ ]:
```
type(b)
```

7.3 Boolean

- Booleans represent one of two values: `True` or `False`.
- In programming you often need to know if an expression is True or False.
- When two values are compared, the expression is evaluated and Python returns the Boolean answer.

[ ]:
```
p = True
q = False
```

[ ]:
```
type(p)
```

7.4 None

- The `None` keyword is used to define a null value, or no value at all.
- None is not the same as 0, False, or an empty string. None is a data type of its own (NoneType) and only None can be None.

[ ]:
```
x = None
```

[ ]:
```
print(x)
```

[ ]:
```
type(x)
```

8 Operators

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Membership Operators
- Identity Operators

8.1 Arithmetic Operators

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

[ ]:
```
a = 18
b = 6
```

8.1.1 Addition +

[ ]:
```
c = a + b
```

[ ]:
```
print(c)
```

[ ]:
```
# Run a operation without print() function directly in jupyter
# will display output below the cell.
a + b
```

[ ]:
```
c
```

[ ]:
```
# If there are multiple lines that give output or return value
# only last line output or return value will be displayed
a + b
a + 2
```

8.1.2 Subtraction -

[ ]:
```
 c = a - b
```

8.1.3 Multiplication *

[ ]:
```
a * b
```

8.1.4 Division /

[ ]:
```
a / b
```

8.1.5 Floor division //

- Returns only the integer part of quotient after division

[ ]:
```
5 // 2
```

8.1.6 Modulus %

- Returns the remainder after the division of one number by another

[ ]:
```
a % 4
```

8.1.7 Exponentiation **

[ ]:
```
a ** 2
```

8.2 Comparison (Relational) Operators

- Equal ==
- Not equal !=
- Less than<
- Less than equal to <=
- Greater than >
- Greater than equal to>=

[ ]:
```
print(a,b)
```

8.2.1 Equal ==

[ ]:
```
a == b
```

8.2.2 Not equal !=

[ ]:
```
a != b
```

8.2.3 Less than<

[ ]:
```
a < b
```

8.2.4 Less than equal to <=

[ ]:
```
a <= 19
```

8.2.5 Greater than >

[ ]:
```
a > b
```

8.2.6 Greater than equal to>=

[ ]:
```
a >= b
```

8.3 Logical Operators

- and
- or
- not

8.3.1 and

```
[ ]: a >= b and a < b
```

```
[ ]: a >= b and a != b
```

8.3.2 or

```
[ ]: a >= b or a < b
```

8.3.3 not

```
[ ]: not (a >= b and a < b)
```

8.4 Membership Operators

- in
- not in

8.4.1 in

```
[ ]: s = "happy"
```

```
[ ]: "py" in s
```

8.4.2 not in

```
[ ]: "A" not in s
```

```
[ ]: "y" not in s
```

9 Operator Precedence

The operator precedence in Python is listed in the following table. It is in descending order (upper group has higher precedence than the lower ones).

Operators	Meaning
()	Parentheses
**	Exponent

Operators	Meaning
*, /, //, %	Multiplication, Division, Floor division, Modulus
+, -	Addition, Subtraction
«, »	Bitwise shift operators
&	Bitwise AND
^	Bitwise XOR
==, !=, >, >=, <, <=, `is`, `is not`, `in`, `not in`	Comparisons, Identity, Membership operators
`not`	Logical NOT
`and`	Logical AND
`or`	Logical OR

10 Associativity of Python Operators

- As in the above table that more than one operator exists in the same group. These operators have the same precedence.
- When two operators have the same precedence, associativity helps to determine the order of operations.
- Associativity is the order in which an expression is evaluated that has multiple operators of the same precedence. Almost all the operators have left-to-right associativity.
- For example, multiplication and floor division have the same precedence. Hence, if both of them are present in an expression, the left one is evaluated first.
- **Non associative operators** Some operators like assignment operators and comparison operators do not have associativity in Python.

11 Type Casting

Converting data types of the variables

[ ]:
```
d = 5
```

[ ]:
```
type(d)
```

11.1 `int` to `float`

[ ]:
```
e = float(d)
```

[ ]:
```
e
```

[ ]:
```
type(e)
```

11.2 float to int

[ ]:
```
int(e)
```

[ ]:
```
e
```

11.3 input gives string, typecast to int or float as per the requirement

[ ]:
```
radius = float(input("Enter Radius:"))
```

[ ]:
```
radius
```

[ ]:
```
type(radius)
```

[ ]:
```

```