Module 1:

Variables, datatypes and I/O

By Ninad Gaikwad

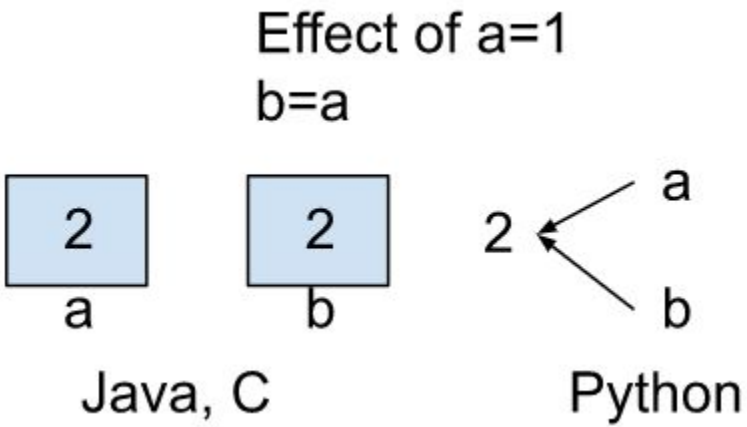
Reference - “Core Python Programming”
Dr. R. Nageshwara Rao
Dreamtech Press

Comments in Python

- Single line comments (#)
- Multi line comments (‘’’ or “”””)
  - Actually not supported by python
  - They are multi line strings not assigned to any variable and hence removed by garbage collector
- Docstrings
  - Description at the beginning of the file
  - Generated into html format by using the pydoc module - “python -m pydoc -w ex” (ex is the name of the python file)

Variables

- Variables in python reference the values
- No separate memory block allocated



Generic Datatypes in python

1. None
2. Numeric
3. Sequences
4. Sets
5. Mappings

Datatypes: None and Numeric

- None type
  Default value for arguments
- Numeric type
  a. Int
     - Any numeric data without decimal points
  b. Float
     - Decimal point values
     - Scientific values such as x=22.55e3 denotes $22.55 * 10^3$
  c. Complex
     - Values in the format a+bj eg. c1= -1-5.5j or –1-5.5J
  d. Representing Binary, Octal. Hexadecimal numbers
     - Binary - “0b110110” or “0B110110”
     - Octal - “0o1724” or “0O1724”
     - Hexadecimal - “0x1af4” or “0X1af4”
  e. Converting Datatypes explicitly
     - Numeric to Numeric: int(x), float(x), complex(x) or complex(x,y)
     - One type to other: oct(a), bin(a), hex(a) Integer to octal, binary and hexadecimal
     - String to numeric: int(str,16) # str to int
  f. bool datatype (true, false)
     - Conditions are evaluated to boolean

Datatypes: Sequences

1. str
   a. Within single (‘a‘) or double quotes (“a”)
   b. Multiline as triple single (‘’’a’’’) or triple double quotes (“””a”””)
2. bytes
   a. Numbers in the range of 0 to 255
   b. e=[10, 20, 0, 40, 50] =>> bytes(e)
   c. Cannot edit or modify elements of bytes type array
3. bytestream
   a. Similar to bytes type array but modification is allowed
   b. Conversion to bytearray: bytearray(e)
4. list
   a. Can store different types of elements eg. list = [10, -20, ‘vijay’, “marry”]
   b. Can grow at runtime
5. tuple
   a. Read only or immutable list enclosed in () eg. list = (10, -20, ‘vijay’, “marry”)
6. range
   a. Sequence of non modifiable numbers
   b. Eg range(10) # generates numbers from 0 to 9 in increments of 1 or
   c. range(1,10,2) # generates numbers from 1 to 9 in increments of 2

Datatypes: Sets and Mappings

- Sets
  a. set Datatype
     - Unordered, mutable set of non duplicate elements enclosed in {}
     - update() method used to add elements eg. s.update([50,60])
     - remove() method used to remove elements eg. s.remove(50)
  b. frozenset Datatype
     - Immutable set
- Mappings
  a. Group of mutable elements in the form of key value pairs
  b. Dictionary eg. d= {10:'kamal', 11:'pranav, 13:'hasini''}
  c. Empty dictionary can be created
  d. d[10]='Haresh' will modify the dictionary

Literals in Python

- Literal is the value assigned to a variable.
- Types: Numeric, Boolean, String
- Numeric: Integer, Float, Hexadecimal, Octal, Binary, Complex
- Boolean: True, False
- String:
  - ‘ or “ or ‘’’ or “””
  - Escape characters

Determining datatype of variables

- type(a) # gives the type of variable
- Everything is an object of some type of class
- Characters are string class type objects
- User defined datatypes: array, class, module
- Identifiers and reserved words
  - Identifiers: name given to variable, function, class etc
  - Reserved words for specific purpose

Naming conventions in python

1. Package, modules, global and module level variables, instance variables, functions, methods : All lower case letters. Multiple letters separated by _
2. Classes: Each word should start with a capital letter. When class represents an exception it should end with word “Error”
3. Method arguments: Instance methods first argument name should be ‘self’. Class methods first argument should be “cls”
4. Non-accessible entities: two underscores before and two underscores after name. Eg. __init__(self) is a function used in class to initialize variables.

Output statements

1. print()
   - Throws the cursor to the next line.
2. print('string')
   - print("This is the first line") # display string
   - print("this is the \n second line") # can include escape characters
   - print("this is the \\n first line") # escape sequence of special characters
   - print("City name ="+"Mumbai") # concatenation character '+'
   - print("City name =","Mumbai") # concatenation character ','
3. print(variable list)
   - print(a,b) # a,b = 2,4 (default separator is space, default end character is \n)
   - print(a,b, sep =":") # o/p => 2:4
   - print(a,b, end ="\t") # adds \t at the end of print statement instead of \n
4. print(object)
   - Lists, tuples, sets or dictionaries can be printed directly
5. print("string", variable list)
   - print(a, ' is a number and so is ', b) # a=3, b=6

Output statements

6. print(formatted string)
   - Using special operators ‘%’
     i. %i or %d represent decimal or integer values, %s for string, %f for float, %c for character
     ii. print(‘x=%i y=%d’ % (x,y)) # z=15, y=43
     iii. print(‘my name is %s’ % name[0:2]) # name=”ninad” will display “ni”
     iv. print(‘val is {%8.2f)’ %num} # will display “val is { 123.45}” contains 8 digits in all out of which one is decimal point two are numbers after decimal rest are digits before decimal including spaces
   - Using replacement fields ‘{ }’
     i. print(“Hello {0}, your salary is {1}”.format(name, salary)) # will print “Hello ravi, your salary is 1000.123” if name=”ravi” salary=1000.123
     ii. print(“Hello {n}, your salary is {s}”.format(n=name, s=salary)) # will print “Hello ravi, your salary is 1000.123”
     iii. print(“Hello {:s}, your salary is {:.2f}”.format(name, salary)) # will print “Hello ravi, your salary is 1000.12” if name=”ravi” salary=1000.123
     iv. print(“Hello %s, your salary is %.2f”.format(name, salary)) # will print “Hello ravi, your salary is 1000.12” if name=”ravi” salary=1000.123

Input Statements

1. Input Statement:
   - float(input(“Enter a number: ”)) # Takes string input and converts it to float
   - int(input(“Enter a number: ”),16) # convert hexadecimal number to decimal similarly for octal and binary
   - a,b = [int(x) for x in input(“Enter two numbers”).split()] # Input two numbers entered by user separated by space
   - lst = [x for x in input(“Enter two numbers”).split(‘,’)] # input list entered by user
2. Evaluation function:
   - result = eval(“a+b-4”) # a, b = 4, 10
   - x = eval(input(‘Input an expression’)) # provide expression from user
3. Command line arguments
   - The default “sys.argv” list
     i. Arguments are stored in the “argv” list of the sys module. Can be accessed by referring to sys.argv
     ii. The first argument “sys.argv[0]” is the name of the program followed by the arguments passed. Arguments can be retrieved as follows
     Eg, args = sys.argv[1:]

Input Statements

- Parsing command line arguments
  i. argparse module is used
  ii. Eg:

```
import argparse

# Create ArgumentParser class object
parser= argparse.ArgumentParser(description= ‘calculates sum of two given numbers’)

# add two arguments with names n1 and n2
parser.add_argument(“n1”, type=float, help=’Input first number’)
parser.add_argument(“n2”, type=float, help=’Input second number’)
# parser.add_argument(‘nums’, nargs=’*’) # accept all arguments
# parser.add_argument(‘nums’, nargs=’+’) # accept 1 or more arguments

# retrieve arguments passed to the program
args = parser.parse_args()

# Convert n1 and n2 to float and add them
result = float(args.n1) + float(args.n2)
print(result)
```