# 204700 – 1/2559

# DATA STRUCTURE AND PROGRAMMING LANGUAGES

# โครงสร้างข้อมูลและภาษาโปรแกรม

*Programming with Python – Part I*

Adapted/Assembled by Areerat Trongratsameethong

# Objectives

- Getting Help
- Syntax
- Arithmetic Expression
- Variables
- Data Types
- String
- Built-In Function
- Input Function
- Flow Control Statement

# Getting Help

- Help in Python is always available right in the interpreter.
- If you want to know how an object works, all you have to do is call
  - **help(<object>)**
- Also useful are
  - **dir()**, which shows you all the object's methods,
  - **<object>.__doc__**, which shows you its documentation string

```
>>> help(5) # Help on int object
>>> dir(5)
['__abs__', '__add__', ...]
>>> abs.__doc__
#'abs(number) -> number Return the absolute value of the argument.'
```

Tutorial - Learn Python in 10 minutes: https://www.stavros.io/tutorials/python/

# Syntax

- **Syntax**
  - Python has no mandatory statement termination characters and blocks are **specified by indentation**.
    - Indent to begin a block,
    - Dedent to end one.
  - Statements that expect an indentation level end in a colon (:).
  - Comments start with the pound (#) sign and are single-line, multi-line strings are used for multi-line comments.
  - Values are assigned (in fact, objects are bound to names) with the equals sign ("="),
  - Equality testing is done using two equals signs ("==").
  - Increment/decrement values using the += and -= operators respectively by the right-hand amount. This works on many datatypes, strings included.
  - Multiple variables can be used on one line.

```
>>> myvar = 3
>>> myvar += 2
>>> myvar
5
>>> myvar -= 1
>>> myvar
4
```

```
"""This is a multiline comment.
The following lines concatenate the two strings."""
```

```
>>> mystring = "Hello"
>>> mystring += " world."
>>> print (mystring)
```

```
>>> myvar = "Hello"
>>> mystring = "World"
>>> myvar, mystring = mystring, myvar
>>> print(myvar + " " + mystring)
```

```
# This swaps the variables in one line(!). It doesn't violate strong typing because
# values aren’t actually being assigned, but new objects are bound to the old names.
```

# Arithmetic Expressions

- **Mathematical Operators**

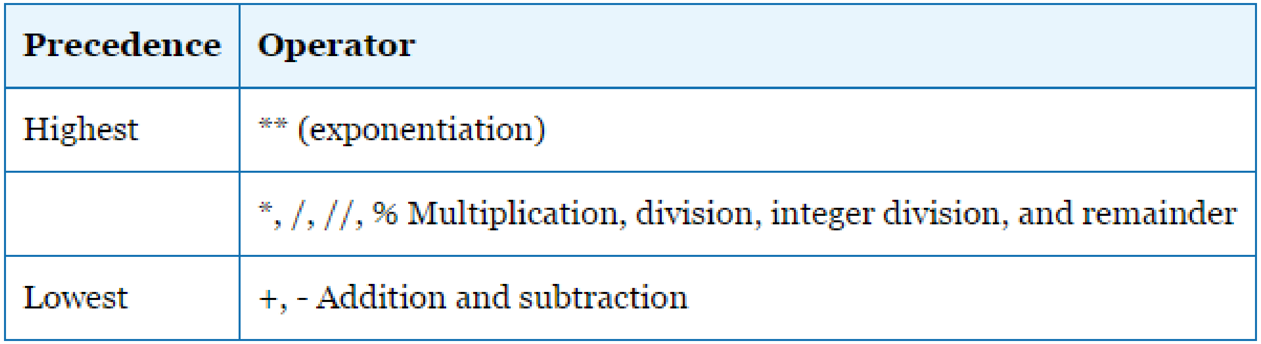
| | | | |
|---|---|---|---|
| `+` | Addition | | |
| `-` | Subtraction | `//` | Integer division |
| `*` | Multiplication | `**` | Exponentiation |
| `/` | Division | `%` | Modulo (remainder) |

- Python is like a calculator: type an expression and it tells you the value.

```
>>> 2 + 3 * 5
17
```

# Arithmetic Expressions

- **Mathematical Operators (Cont.)**
- **Order of Evaluation**



| Precedence | Operator |
|---|---|
| Highest | ** (exponentiation) |
| | *, /, //, % Multiplication, division, integer division, and remainder |
| Lowest | +, - Addition and subtraction |

**For more Python Operator Precedence:**

**http://www.mathcs.emory.edu/~valerie/courses/fall10/155/resources/op_precedence.html**

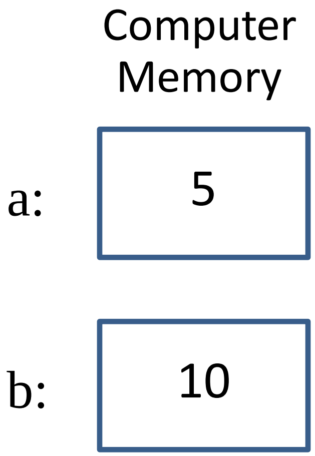
- **Use parentheses to force alternate precedence**

```
5 * 6 + 7       ≠ 5 * (6 + 7)
5 * 10 % 4      = (5 * 10) % 4
2 + 3 + 4       = (2 + 3) + 4
2 ** 3 ** 4     = 2 ** (3 ** 4)
```

# Variables

- In Python programming, a variable is a name you give a value.
- In Python we give a name to a value using an ***assignment statement****:*

```
>>> a = 5       # Assignment Statement
>>> a           # Expression
5              # Python's Response
>> > b = 2 * a
>>> b
10
```



Computer Memory

a: 5

b: 10

# Variables

- Variable Name
  - All variable names must start with a letter (lowercase recommended).
  - The remainder of the variable name (if any) can consist of any combination of uppercase letters, lowercase letters, digits and underscores (_).
  - Identifiers in Python are case sensitive.
    - **Example**: **Value is not** the same as **value** .

# Data Types

- **Data Types**
  - Integers (int)
  - Floating Point (float)
  - String (str)
  - Boolean (bool)

  For more information, please see: https://docs.python.org/3.1/library/stdtypes.html

```
- Integer Division in Python3:

    7 / 2        equals 3.5
    7 // 2       equals 3
    7 // 2.0     equals 3.0
    7.0 // 2     equals 3.0
    -7 // 2      equals -4

- Beware! // rounds down to smaller number, not towards zero
```

# Data Types

- **Data Types (Cont.)**
- The data structures available in python are
  - **Lists**, **Tuples** and **Dictionaries**
    - **Lists** are like one-dimensional arrays (but you can also have lists of other lists)
    - **Dictionaries** are associative arrays (a.k.a. hash tables)
    - **Tuples** are **immutable one-dimensional arrays**
  - Python "**arrays**" can be of any type, so you can mix e.g. integers, strings, etc in lists/dictionaries/tuples.
    - The index of the first item in all array types is 0.
    - Negative numbers count from the end towards the beginning, -1 is the last item.
    - Variables can point to functions.
  - Sets are available in the sets library (but are built-in in Python 2.5 and later).

```
>>> sample = [1, ["another", "list"], ("a", "tuple")]
>>> mylist = ["List item 1", 2, 3.14]
>>> mylist[0] = "List item 1 again" # We're changing the item.
>>> mylist[-1] = 3.21 # Here, we refer to the last item.
>>> mydict = {"Key 1": "Value 1", 2: 3, "pi": 3.14}
>>> mydict["pi"] = 3.15 # This is how you change dictionary values.
>>> mytuple = (1, 2, 3)
>>> myfunction = len
>>> print (myfunction(mylist))
3
```

# Data Types

- **Data Types (Cont.)**
- We can access array ranges using a colon (:)
  - Leaving the start index empty assumes the first item,
  - Leaving the end index assumes the last item.
  - Negative indexes count from the last item backwards (thus -1 is the last item)

```
>>> mylist = ["List item 1", 2, 3.14]
>>> print (mylist[:])
['List item 1', 2, 3.1400000000000001]
>>> print (mylist[0:2])
['List item 1', 2]
>>> print (mylist[-3:-1])
['List item 1', 2]
>>> print (mylist[1:])
[2, 3.14]
# Adding a third parameter, "step" will have Python step in
# N item increments, rather than 1.
# E.g., this will return the first item, then go to the third and
# return that (so, items 0 and 2 in 0-indexing).
>>> print mylist[::2]
['List item 1', 3.14]
```

# String

- **String**
- Strings can use either single or double quotation marks, and you can have quotation marks of one kind inside a string that uses the other kind (i.e. "He said 'hello'." is valid).
- Multiline strings are enclosed in triple double (or single) quotes (""").
- Python supports Unicode out of the box, using the syntax u"This is a unicode string".
- To fill a string with values:
  - Use the % operator and a tuple.
  - Each %s gets replaced with an item from the tuple, left to right, and you can also use dictionary substitutions.

```
>>> print("Name: %s\nNumber: %s \nString: %s" % ("Type your name here", 3, 3 * "-"))
Name: Type your name here
Number: 3
String: ---

>>> strString = """This is
a multiline string."""
>>> print(strString)
This is
a multiline string.

>>> print ("This %(verb)s a %(noun)s." % {"noun": "test", "verb": "is"})
This is a test.
```

```
>>> strString = "He said 'hello'."
>>> print(strString)
He said 'hello'.
```

# Built-In Functions

- **Lots of math stuff, e.g., sqrt, log, sin, cos**
- **math** is a predefined module of functions (also called methods) that we can use without writing their implementations.

```
import math
r = 5 + math.sqrt(2)
alpha = math.sin(math.pi/3)
```

For more details, please see: https://docs.python.org/3/library/functions.html

# Input Function

- **Input** can come in various ways, for example from
  - database,
  - another computer,
  - mouse clicks
  - **Keyboard:** Python provides the function input(). input has an optional parameter, which is the prompt string.

**Example of Input Function**

```
>>> person = input('Enter your name: ')
Enter your name: Jack
>>> print("Hello", person)
Hello Jack
>>> print('Hello ', person, '!', sep=' ') # sep = separater
Hello  Jack !
>>> print('Hello ', person, '!', sep='|')
Hello |Jack|!
```

For more details, please see: https://docs.python.org/3/library/functions.html

# Flow Control Statement

- **Flow control statements are: If, for,** and **while**

- **Simple Condition and if Statement**

**Simple Conditions**

```
print(2 < 5)          # True
print(3 > 7)          # False
x = 11
print(x > 10)         # True
print(2 * x < x)      # False
print(type(True)) # <class 'bool'>
```

**The general Python syntax for a simple if statement is**

```
if condition :
    indentedStatementBlock>
```

If the condition is true, then do the indented statements. If the condition is *not true*, then *skip the indented statements*.

**Example of Simple if Statements**

```
weight = float(input("How many pounds does your suitcase weigh? "))
if weight > 50:
    print("There is a $25 charge for luggage that heavy.")
print("Thank you for your business.")
```

**Another Example of Simple if Statement**

```
if balance < 0:
    transfer = -balance
    # transfer enough from the backup account:
    backupAccount = backupAccount - transfer
    balance = balance + transfer
```

**Reference**: http://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/ifstatements.html

# Flow Control Statement

- **Flow control statements (Cont.)**
- **if-else** Statement

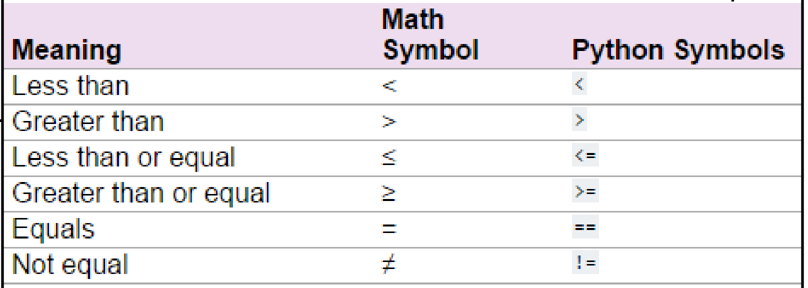
The general Python **if-else** syntax is

```
if condition :
    indentedStatementBlockForTrueCondition
else:
    indentedStatementBlockForFalseConditionif-else Statements
```

**Example of if-else Statements**

```
temperature = float(input('What is the temperature? '))
if temperature > 70:
    print('Wear shorts.')
else:
    print('Wear long pants.')
print('Get some exercise outside.')
```

**More Conditional Expressions**



| Meaning | Math Symbol | Python Symbols |
|---|---|---|
| Less than | < | < |
| Greater than | > | > |
| Less than or equal | ≤ | <= |
| Greater than or equal | ≥ | >= |
| Equals | = | == |
| Not equal | ≠ | != |

# Flow Control Statement

- **Flow control statements (Cont.)**
- **for** Statement

```
>>> words = ['cat', 'window', 'defenestrate']
>>> for w in words:
        print(w, len(w))


cat 3
window 6
defenestrate 12
```

```
>>> for w in words[:]:  # Loop over a slice copy of the entire list.
        if len(w) > 6:
                words.insert(0, w)


>>> words
['defenestrate', 'cat', 'window', 'defenestrate']
>>>
```

**Reference:** https://docs.python.org/3/tutorial/controlflow.html

# Flow Control Statement

- **Flow control statements (Cont.)**

- **For Statement:** the **range()** Function

```
>>> for i in range(5):
        print(i)

0
1
2
3
4
```

```
>>> list(range(5))
[0, 1, 2, 3, 4]
>>> print(list(range(5)))
[0, 1, 2, 3, 4]
```

```
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
>>> for i in range(len(a)):
        print(i, a[i])

0 Mary
1 had
2 a
3 little
4 lamb
```

# Flow Control Statement

- **Flow control statements (Cont.)**
- **break** and continue Statements, and **else Clauses on Loops**

```
>>> for n in range(2, 10):
        for x in range(2, n):
            if n % x == 0:
                print(n, 'equals', x, '*', n//x)
                break
        else:
            # loop fell through without finding a factor
            print(n, 'is a prime number')

2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

# Flow Control Statement

- **Flow control statements (Cont.)**
- break and **continue** Statements, and else Clauses on Loops

```
>>> for num in range(2, 10):
        if num % 2 == 0:
            print("Found an even number", num)
            continue
        print("Found a number", num)


Found an even number 2
Found a number 3
Found an even number 4
Found a number 5
Found an even number 6
Found a number 7
Found an even number 8
Found a number 9
```

# Flow Control Statement

- **Flow control statements (Cont.)**
- **pass** Statements: The pass statement **does nothing**. It can be used when a statement is required syntactically but the program requires no action.

```
rangelist = range(10)
print (rangelist)

for number in rangelist:
    # Check if number is one of the numbers in the tuple.
    if number in (3, 4, 7, 9):
        # "Break" terminates a for without executing the "else" clause.
        break
    else:
        # "Continue" starts the next iteration of the loop.
        # It's rather useless here, as it's the last statement of the loop.
        continue
else:
    # The "else" clause is optional and is executed only if the loop didn't "break".
    pass # Do nothing

if rangelist[1] == 2:
    print ("The second item (lists are 0-based) is 2")
elif rangelist[1] == 3:
    print ("The second item (lists are 0-based) is 3")
else:
    print ("Dunno")

while rangelist[1] == 1:
    pass
```

# Flow Control Statement

- **Flow control statements (Cont.)**
- **while** Statement

A while loop generally follows the pattern of the successive modification loop introduced with for-each loops:

```
initialization
while continuationCondition :
    do main action to be repeated
    prepare variables for the next time through the loop
```

**Example of while loop**

```
# Prints out 0,1,2,3,4

count = 0
while count < 5:
    print (count)
    count += 1  # This is the same as count = count + 1
```