Data and File Structures Laboratory

Introduction to Python

Malay Bhattacharyya

Assistant Professor

Machine Intelligence Unit
Indian Statistical Institute, Kolkata

November, 2019

The Python interpreter

Source code $\rightarrow$ **(Python interpreter)** $\rightarrow$ **Executable**

Standard versions of Python

Significant Features	Python 2	Python 3
`print`	As a statement	As a function
`xrange()`	Yes	No
Returning lists	Yes	No
Returning iterable objects	No	Yes
Unicode	No	Yes
`byte` type	No	Yes
Exception handling with `as`	No	Yes
Integer division	Traditional	New

Note: Python 1 is no more in use and Python 2 is soon to be obsolete.

The first Python program (in Python 3)

Source: Welcome2Python.py

The first Python program (in Python 3)

Source: Welcome2Python.py

```
print("Welcome 2 Python")
```

The first Python program (in Python 3)

Source: Welcome2Python.py

```
print("Welcome 2 Python")
```

Execution: python3 Welcome2Python.py

The first Python program (in Python 3)

Source: Welcome2Python.py

```
print("Welcome 2 Python")
```

Execution: python3 Welcome2Python.py

Welcome 2 Python**(cursor here!!!)**

Dissecting the code

```
# Import Statements
import math
# Function Definitions
def div(a, b):
    return a/b # Note the indentation
# Statements
var1 = 3
var2 = 2
# Functions
division = div(var1, var2) # Function call
print(division) # Prints 1.5
print(not (division > math.pi)) # Prints True
```

Dissecting the code

```
# Import Statements
import math
# Function Definitions
def div(a, b):
    return a/b # Note the indentation
# Statements
var1 = 3
var2 = 2
# Functions
division = div(var1, var2) # Function call
print(division) # Prints 1.5
print(not (division > math.pi)) # Prints True
```

Note: The program name can be anything.

Basic characteristics

- Python is a free and open source language.

Basic characteristics

- Python is a free and open source language.
- Python is an interpreted language.

Basic characteristics

- Python is a free and open source language.
- Python is an interpreted language.
- Python is not a free-form language.

Basic characteristics

- Python is a free and open source language.
- Python is an interpreted language.
- Python is not a free-form language.
- Python is a strongly typed language.

Basic characteristics

- Python is a free and open source language.
- Python is an interpreted language.
- Python is not a free-form language.
- Python is a strongly typed language.
- Python is an object-oriented language but it also supports procedural oriented programming.

Basic characteristics

- Python is a free and open source language.
- Python is an interpreted language.
- Python is not a free-form language.
- Python is a strongly typed language.
- Python is an object-oriented language but it also supports procedural oriented programming.
- Python is a high level language.

Basic characteristics

- Python is a free and open source language.
- Python is an interpreted language.
- Python is not a free-form language.
- Python is a strongly typed language.
- Python is an object-oriented language but it also supports procedural oriented programming.
- Python is a high level language.
- Python is a portable and cross-platform language.

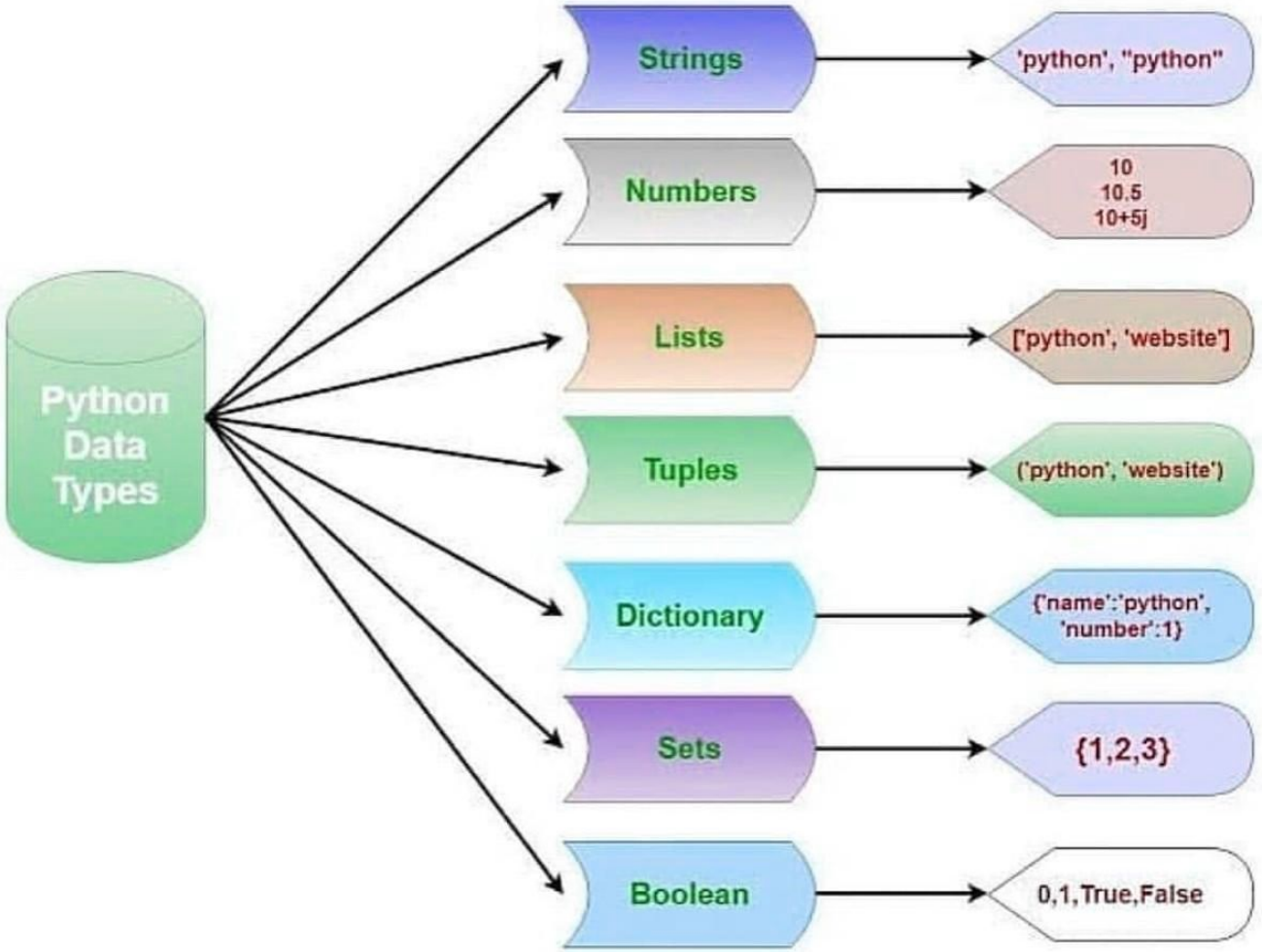
Basic characteristics

- Python is a free and open source language.
- Python is an interpreted language.
- Python is not a free-form language.
- Python is a strongly typed language.
- Python is an object-oriented language but it also supports procedural oriented programming.
- Python is a high level language.
- Python is a portable and cross-platform language.
- Python is an extensible language.

Basic characteristics

- Python is a free and open source language.
- Python is an interpreted language.
- Python is not a free-form language.
- Python is a strongly typed language.
- Python is an object-oriented language but it also supports procedural oriented programming.
- Python is a high level language.
- Python is a portable and cross-platform language.
- Python is an extensible language.

The data types in Python



Numbers

Real values including integers

Examples:

```
1.2345    3.45e67
1.        +3.45e67
.1        -3.45e-67
-0.6789   .00345e-32
+.4560    1e-15
-.1234    1e+15
```

Numbers

Real values including integers

Examples:

```
1.2345      3.45e67
1.          +3.45e67
.1          -3.45e-67
-0.6789     .00345e-32
+.4560      1e-15
-.1234      1e+15
```

- Do not use commas as thousand-separators.
- At times behavior may be counter-intuitive.

Boolean values

Any non-zero value is treated as `TRUE` and zero is treated as `FALSE`.

Examples:

0	False	0e10	False
1	True	'A'	True
6 - 2 * 3	False	"A"	True
(6 - 2) * 3	True	'\0'	True
0.0075	True	(0, 0)	True

Boolean values

Any non-zero value is treated as `TRUE` and zero is treated as `FALSE`.

Examples:

0	False	0e10	False
1	True	'A'	True
6 - 2 * 3	False	"A"	True
(6 - 2) * 3	True	'\0'	True
0.0075	True	(0, 0)	True

Note: The expressions like "x = 0" or "x = 1" will exhibit error.

Mutable and immutable objects

Everything in Python is an object and it is either mutable or immutable.

Mutable and immutable objects

Everything in Python is an object and it is either mutable or immutable.

A mutable object can be changed to a different type after it is created, but an immutable object cannot be changed.

Mutable and immutable objects

Everything in Python is an object and it is either mutable or immutable.

A mutable object can be changed to a different type after it is created, but an immutable object cannot be changed.

- Objects of built-in types like int, float, bool, str, tuple, unicode are immutable
- Objects of built-in types like list, set, dict are mutable.

Arithmetic operators

Summation $(+)$

Subtraction $(-)$

Multiplication $(*)$

Power $(**)$

Float Division $(/)$

Floor Division $(//)$

Modulo division $(\%)$

Arithmetic operators

Summation $(+)$

Subtraction $(-)$

Multiplication $(*)$

Power $(**)$

Float Division $(/)$

Floor Division $(//)$

Modulo division $(\%)$

Note: Modulo division operator works on any type of numbers (including floating point values and negatives!!!) and returns the sign of the denominator.

Modulo division on Boolean values

What will be the output of the following program?

```
var1 = True
var2 = False
print(var1 % var1)
print(var1 % var2)
print(var2 % var1)
print(var2 % var2)
```

Modulo division on Boolean values

What will be the output of the following program?

```
var1 = True
var2 = False
print(var1 % var1)
print(var1 % var2)
print(var2 % var1)
print(var2 % var2)
```

0
Error
0
Error

Relational operators

Less than $(<)$

Less than equals to $(<=)$

Greater than $(>)$

Greater than equals to $(>=)$

Equals to $(==)$

Not equals to $(!=)$

Logical operators

Logical and (`and`)

Logical or (`or`)

Logical not (`not`)

Assignment operators

Assignment $(=)$

Addition and assignment $(+=)$

Subtraction and assignment $(-=)$

Multiplication and assignment $(*=)$

Power and assignment $(**=)$

Float Division and assignment $(/=)$

Floor Division and assignment $(//=)$

Modulo division and assignment $(\%=)$

Assignment operators

What will be the output of the following program?

```
a = 10
b = 20
a, b = b, a # Swapping values
print(a)
print(b)
```

Assignment operators

What will be the output of the following program?

```
a = 10
b = 20
a, b = b, a # Swapping values
print(a)
print(b)
```

20
10

Bitwise operators

Bitwise and $(\&)$

Bitwise or $(|)$

Bitwise not $(\sim)$

Bitwise xor $(\wedge)$

Left shift $(<<)$

Right shift $(>>)$

Identity operators

Identical (`is`)

Not identical (`is not`)

Identity operators

Identical (`is`)

Not identical (`is not`)

Note: Two variables that are equal does not imply that they are identical. For being identical, they must be located on the same part of the memory.

Identity operators

What will be the output of the following program?

```
var1 = 123
var2 = 123
print(var1 is var2)
var1 = 'Python'
var2 = 'Python'
print(var1 is not var2)
var1 = [1, 2, 3] # List of immutable objects
var2 = [1, 2, 3] # List of immutable objects
print(var1 is var2)
```

Identity operators

What will be the output of the following program?

```
var1 = 123
var2 = 123
print(var1 is var2)
var1 = 'Python'
var2 = 'Python'
print(var1 is not var2)
var1 = [1, 2, 3] # List of immutable objects
var2 = [1, 2, 3] # List of immutable objects
print(var1 is var2)
```

True
False
False

Some comments

The following popular operators are not available in Python:

- **Increment** (++)
- **Decrement** (- -)
- **Comma** (,)

Use of comma

What will be the output of the following program?

```
var = 20, 30, 40
print(var)
var = (50, 60, 70)
print(var)
```

Use of comma

What will be the output of the following program?

```
var = 20, 30, 40
print(var)
var = (50, 60, 70)
print(var)
```

(20, 30, 40)
(50, 60, 70)

Conditional – if-else

```
if <Condition>:
    statement 1
    statement 2
else:
    statement 3 # Execute if Condition fails
```

Iterative – if-elif-else

```
if <Condition 1>:
    statement 1
elif <Condition 2>:
    Statement 2
else:
    statement 3 # Execute if Condition 1 and 2 fails
```

Iterative – for loop

```
for <variable> in <container>:
    statement 1
    statement 2
```

Iterative – for loop

We can create a list of consecutive integers using the `range()` function as follows.

```
for <variable> in range(<value>):
    statement 1
    statement 2
```

- `range(x)` returns a list whose items are consecutive integers from [0, x).
- `range(x, y)` returns a list (feasible when x $<$ y) whose items are consecutive integers from [x, y).
- `range(x, y, step)` returns a list of integers from [x, y), such that the difference between each two adjacent items in the list is step. If step is less than 0, it counts down from x to y. If step equals 0, it raises an exception.

Iterative – while loop

```
while <Condition>:
    statement 1
    statement 2
```

break and continue

- **break:** Immediately jump to the next operation after the loop
- **continue:** Do the operation, if applicable, and continue with the next iteration of the loop

break and continue

- **break:** Immediately jump to the next operation after the loop
- **continue:** Do the operation, if applicable, and continue with the next iteration of the loop

```
for i in range(1, 100):
    print(i)
    if i%10 != 0:
        break
```

Prints only 1

break and continue

- **break:** Immediately jump to the next operation after the loop
- **continue:** Do the operation, if applicable, and continue with the next iteration of the loop

```
for i in range(1, 100):
    print(i)
    if i%10 != 0:
        break
```

Prints only 1

```
i = 0
while i < 100:
    i = i+1
    if i%10 != 0:
        continue
    print(i)
```

Prints 10, 20, ..., 100

Input/Output

I/O from the terminal:

- `print()` # Value can be printed without mentioning the type
- `input()` # Value is taken in a string and can be converted to appropriate type using int(), float(), bool(), etc.

I/O from files:

- `open()`, `close()` # Files are opened in r/w/a mode and the address is returned to a file pointer
- `read()`, `write()` # With a file pointer
- `readline()` # With a file pointer
- `readlines()`, `writelines()` # With a file pointer

Functions

```
def <function-name>(<argument 1>, ..., <argument n>):
    Statement 1
    Statement 2
    Statement 3
```

Functions

```
def <function-name>(<argument 1>, ..., <argument n>):
    Statement 1
    Statement 2
    Statement 3
```

Note: You may either return a `<value>` or return nothing based on your requirement.

Problems – Day 25

1. Take n integers $\{a_i\}$ as user inputs, and compute $\sum_{i=1}^{n} a_i$. How long does it take with naive repeated additions? Can you perform better with an efficient approach?
2. The `Heron triangles` are those special triangles that have integer sides and integer area. Take the three vertices of any arbitrary triangle (say $(x1, y1)$, $(x2, y2)$ and $(x3, y3)$) from the user as inputs, and determine whether they form the sides of a Heron triangle or not.
3. Write a program to print the following diamond-like pattern using generic controls over the print. Let the line number be user input.

```
  *
 ***
*****
 ***
  *
```

Problems – Day 25

4. Suppose m and n are (signed) integers and x and y are floating variables. Write logical conditions evaluating to `True` iff:
   - $x + y$ is an integer.
   - m lies strictly between x and y.
   - m equals the integer part of x.
   - x is positive with integer part at least 3 and with fractional part less than 0.3.
   - m and n have the same parity (i.e., are both odd or both even).
   - m is a perfect square.
5. Take a pair of points from the user (say $(x1, y1)$ and $(x2, y2)$) and construct a path joining them together. Run a loop to take k more points from the user, one at a time, and determine whether there was a left turn (L), a right turn (R) or a straight walk (S) in relation to the previous two points.
   Sample Input: (0, 0) (2, 2) (3, 2) (4, 2) (5, 1) (6, 4)
   Sample Output: - - R S R L

Problems – Day 25

6. Write a program that takes text typed at the terminal as user input, and counts the following:
   (i) The number of occurrences of vowels in the input text.
   (ii) The number of words in the input text. Assume that any contiguous sequence of letters and / or digits forms one word.
7. Write a program that will print its own source code on the terminal as output.
8. Write a program that will take a string as user input and generate all of its prefixes.