Standard Data Types

Harish D. Gadade
www.harishgadade.com

Standard Data Type

1. Numeric
2. Boolean
3. String
4. List
5. Tuples
6. Dictionary
7. Sets

1. Numeric

- Numeric data comes in two flavours
  - Int - Integer
  - Float - Fractional Numbers
- 150, -5, 564123 are values of integer type
- 10.52, -0.01, 25.23156 are values of type float

1. Numeric

- Operations on Numbers
  - Normal arithmetic operations : `+, -, *, /`
  - Note that, / always produces a float
  - Quotient and Remainder : `// and %`
  - Exponentiation : `**`
- Other Operations on Numbers
  - `log(), sqrt(), sin(),.....`
  - Built-in python but not available by default
  - Must include “math library”
  - `from math import *`

1. Numeric

- Operations on Numbers
  - Normal arithmetic operations : **+, -, *, /**
  - Note that, / always produces a float
  - Quotient and Remainder : **// and %**
  - Exponentiation : ******
- Other Operations on Numbers
  - **log(), sqrt(), sin(),.....**
  - Built-in python but not available by default
  - Must include “math library”
  - **from math import ***

```
>>> 5+8
13
>>> 5/3
1.6666666666666667
>>> 5//2
2
>>> 5%2
1
>>> 5**2
25
```

1. Numeric

- Operations on Numbers
  - Normal arithmetic operations : **+, -, *, /**
  - Note that, / always produces a float
  - Quotient and Remainder : **// and %**
  - Exponentiation : ******
- Other Operations on Numbers
  - **log(), sqrt(), sin(),.....**
  - Built-in python but not available by default
  - Must include “math library”
  - **from math import ***

```
>>> from math import*
>>> sqrt(4)
2.0
>>> sin(90)
0.8939966636005579
>>> log(2)
0.6931471805599453
>>>
```

2. Boolean Values : bool

- True, False
- Logical Operators : nor, and, or
  - `Not True` is False, `Not False` is True
  - x `and` y is True, if both of x, y are True
  - x `or` y is True, if atleast one of x,y is True

```
>>> a=True
>>> type(a)
<class
'bool'>
>>> x=False
>>> type(x)
<class
'bool'>
```

3. String (str)

- Strings are another important data type in Python.
- Type string is a sequence of character
  - A single character is a string of length 1
  - No separate type char
- Enclose in quotes - Single, double, triple

```
City = 'Pune'
Class = " It's Harish Gadade's Class"
Name = ''' It's  name "Python Programming" '''
```

3. String (str)

- **`String`** is a sequence of characters
- Position 0,1,2,..... N-1 for a string of length n
  - s = “ Python”

0	1	2	3	4	5
P	y	t	h	o	n
-6	-5	-4	-3	-2	-1

- Positions -1,-2, ... count backwards from end
- There are several operators such as `slicing [ : ], concatenation ( + ) and repetition ( * )`

3. String (str)

- **`String`** is a sequence of characters
- Position 0,1,2,..... N-1 for a string of length n
  - s = “ Python”

0	1	2	3	4	5
P	y	t	h	o	n
-6	-5	-4	-3	-2	-1

- Strings are Immutable

- Positions -1,-2, ... count backwards from end
- There are several operators such as `slicing [ : ], concatenation ( + ) and repetition ( * )`

4. List

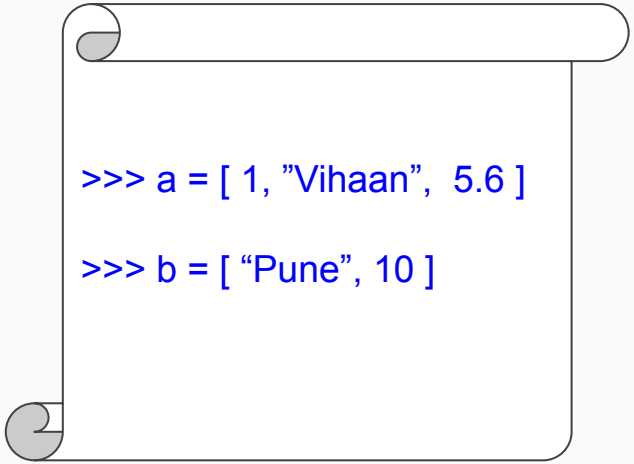
- A `List` can contain same types of Items. Alternatively, a `List` can also contain different types of items.
- To declare a `List`, we need to separate the items using commas and enclose them within a square bracket(`[]`)
- Similar to string data type, list also has `+`, `*` and slicing `[:]` operators for concatenation, repetition and sublist respectively.
- `List` is Mutable

4. List

- Basic Operations
  - Display List
  - Concatenation
  - Repetition
  - Sublist

4. List

- Basic Operations
  - Display List
  - Concatenation
  - Repetition
  - Sublist



4. List

- Basic Operations
  - Display List
  - Concatenation
  - Repetition
  - Sublist

```
>>> a = [ 1, ”Vihaan”,  5.6 ]

>>> b = [ “Pune”, 10 ]
```

Displaying a List

```
>>> a

a = [ 1, ”Vihaan”,  5.6 ]

>>> b

 [ “Pune”, 10 ]
```

4. List

- Basic Operations
  - Display List
  - Concatenation
  - Repetition
  - Sublist

```
>>> a = [ 1, ”Vihaan”,  5.6 ]

>>> b = [ “Pune”, 10 ]
```

Concatenation

```
>>> a + b

a = [ 1, ”Vihaan”,  5.6, “Pune”, 10 ]
```

4. List

- Basic Operations
  - Display List
  - Concatenation
  - Repetition
  - Sublist

```
>>> a = [ 1, ”Vihaan”,  5.6 ]

>>> b = [ “Pune”, 10 ]
```

Repetition

```
>>> b * 3

[ “Pune”, 10 , “Pune”, 10 , “Pune”, 10 ]
```

4. List

- Basic Operations
  - Display List
  - Concatenation
  - Repetition
  - Sublist

```
>>> a = [ 1, ”Vihaan”,  5.6 ]

>>> b = [ “Pune”, 10 ]
```

Sub-List

```
>>>a [ 0 : 1 ]

[1]

>>> a [ 1 : 3 ]

[“Vihaan”, 5.6]
```

5. Tuple

- Similar to list, a **tuple** is also used to store sequence of items.
- Like a list, a tuple consists of items separated by commas.
- However, tuples are enclosed within parentheses rather than square bracket.
- Difference between List and Tuples
  - In List, items are enclosed within square brackets **[ ]** whereas in tuples, items are enclosed within parentheses **()**
  - List are **mutable** whereas Tuples are **immutable.** Tuples are **read only** lists.

5. Tuple

- Examples

```
>>> a=(10,"Vihaan",5.6,"Jalgaon")
>>> a
(10, 'Vihaan', 5.6, 'Jalgaon')
>>> a[1]="Rituja"
Traceback (most recent call last):
File "<pyshell#4>", line 1, in
<module> a[1]="Rituja"
TypeError: 'tuple' object does not
support item assignment
```

6. Dictionary

- **Dictionary** is an unordered collection of key-value pairs.
- The order of elements in a dictionary is undefined but we can iterate over the following:
  - The Key
  - The Value
  - The item (key - Value pairs) in a dictionary
- Items are enclosed in a curly-braces { } and separated by comma (,).
- A colon (:) is used to separate key from value.
- A key inside the square bracket [ ] is used to access the dictionary items
- **Dictionary** values are **mutable**

6. Dictionary

- Example:

```
>>> dict={1:"Jalgaon","two":"Pune"}
>>> dict
{1: 'Jalgaon', 'two': 'Pune'}
>>> dict[3]="Mumbai"
>>> dict
{1: 'Jalgaon', 'two': 'Pune', 3: 'Mumbai'}
```

6. Dictionary

- Example:

```
>>> dict={1:"Jalgaon","two":"Pune"}
>>> dict
{1: 'Jalgaon', 'two': 'Pune'}
>>> dict[3]="Mumbai"
>>> dict
{1: 'Jalgaon', 'two': 'Pune', 3: 'Mumbai'}
```

```
>>> dict.keys()
dict_keys([1, 'two', 3])
>>> dict.values()
dict_values(['Jalgaon', 'Pune',
'Mumbai'])
>>>
```

7. Sets

- An unordered collection of data is known as set.
- A set does not contain duplicate values or elements and it is non-subscriptable
- Union, intersection, difference and symmetric difference are the some operations which can performed on sets.
  - **Union:** All elements from two sets. Operator used is |
  - **Intersection:** Display common elements in two sets. Operator used is **&**
  - **Difference:** Display elements which are present in first set not in other set. Operator used is **-**
  - **Symmetric Difference:** returns elements which are present in either set but not in both. Operator used is **^**

7. Sets

- Example:

```
>>> a = set ( [1,2,3,1,2,8,5,4] )
>>> b = set ( [1,9,3,2,5] )
>>> a
{1, 2, 3, 4, 5, 8}
>>> b
{1, 2, 3, 5, 9}
```

```
>>> intersection = a & b
>>> intersection
{1, 2, 3, 5}
>>> union = a | b
>>> union
{1, 2, 3, 4, 5, 8, 9}
>>>
```

7. Sets

- Example:

```
>>> a = set ( [1,2,3,1,2,8,5,4] )
>>> b = set ( [1,9,3,2,5] )
>>> a
{1, 2, 3, 4, 5, 8}
>>> b
{1, 2, 3, 5, 9}
```

```
>>> diff = a - b
>>> diff
{8, 4}
>>> symm_diff = a ^ b
>>> symm_diff
{4, 8, 9}
```

Type( ) Function

- **`type()`** function is a built-in function which returns the datatype of any arbitrary object.
- **`type()`** function can take anything as an argument and returns its data type such as integer, strings, dictionaries, lists, classes, modules, tuples, function etc

Type( ) Function

- **`type()`** function is a built-in function which returns the datatype of any arbitrary object.
- **`type()`** function can take anything as an argument and returns its data type such as integer, strings, dictionaries, lists, classes, modules, tuples, function etc

```
>>> x=10
>>> type(x)
<class 'int'>
```

```
>>> import os
>>> type(os)
<class 'module'>
```

```
>>> type("Hello")
<class 'str'>
```

```
>>> tup=(1,2,3)
>>> type(tup)
<class 'tuple'>
```

```
>>> lst=[1,2,3]
>>> type(lst)
<class 'list'>
```