02-Python Basic

Data types, dynamic and strong type

Posted on September 6, 2021
Last updated on January 30, 2023

2.1 Data Types

Data Type represent the `type of data present` inside a variable. In Python we are not required to specify the type explicitly.

Based on value provided, the type will be assigned automatically. Hence Python is `Dynamically Typed` Language.

Python contains the following inbuilt data types

```
* Char Type:        str
* Numeric Types:    int, float, complex
* Sequence Types:   list, tuple, range
* Mapping Type:     dict
* Set Types:        set,  frozenset
* Boolean Type:     bool
* Binary Types:     bytes, bytearray, memoryview
```

2.2 Python contains several inbuilt functions

- `type()`

    - to check the type of variable
- `id()`
    - to get address of object
- `print()`
    - to print the value

In Python everything is objec

2.3 Dynamic type and Strong Type

Dynamic Type

- Python is `dynamic typing` language

> - *We don't specify types in advance, the object types is `resolved at runtime`*
> - *Dynamic typing allows for flexibility in programming, but* ***with a price in performance*** *.*

Duck Type

- Also called as `Duck typing` language

> - *Duck typing means that we are not interested in what type an object is*
> - *Instead, we are more concerned in the functional aspect of the object*

`If a bird that walks like a duck and swims like a duck and quacks like a duck, that bird is a duck`

Strong Type

- Python uses `strong typing`

> - *Every object has a specific type*
> - *There is `no implicit type conversion`*
> - *9 + '9' –> Error in python*

2.5 int data type

We can use int data type to `represent whole numbers` (integral values)

In Python2 we have `long data type` to represent very large integral values. But in Python3 there is no long type explicitly and we can represent long values also by using int type only.

2.5.1 Represent different Number System

1. Decimal form(base-10)

- It is the `default number system` in Python
- The allowed digits are: `0 to 9`

```
a = 10
```

2. Binary form(Base-2)

- The allowed digits are : `0 & 1`
- Literal value should be `prefixed with 0b or 0B`

```
a = 0B1111
a = 0B123
a = 0b111
```

3. Octal Form(Base-8)

- The allowed digits are : `0 to 7`
- Literal value should be prefixed with `0o or 0O`

```
a = 0o123
a = 0o786
```

4. Hexa Decimal Form(Base-16)

- The allowed digits are : `0 to 9, a-f` (both lower and upper cases are allowed)
- Literal value should be prefixed with `0x or 0X`

```
a = 0XFACE
a = 0XBeer
```

Note:

- Being a programmer we can specify literal values in decimal, binary, octal and hexa decimal forms.
- But PVM will always provide values only in decimal form.

```
print(10)   # 10
print(0o10) # 8
print(0X10) # 16
print(0B10) # 2
```

2.5.2 Base Conversions

Python provide the following in-built functions for base conversions

1. `bin()`

- We can use bin() to convert from any base to binary

```
bin(15)  # '0b1111'
bin(0o11)  #'0b1001'
bin(0X10) # '0b10000'
```

2. `oct()`

- We can use oct() to convert from any base to octal

```
oct(10)  # '0o12'
oct(0B1111)  # '0o17'
oct(0X123)  # '0o443'
```

3. `hex()`

- We can use hex() to convert from any base to hexa decimal

```
hex(100)  # '0x64'
hex(0B111111)  # '0x3f'
hex(0o12345)  # '0x14e5'
```

2.6 float data type

We can use float data type to `represent floating point values` (decimal values)

We can also represent floating point values by using `exponential form` (scientific notation)

We can represent float values only by using decimal form.

```
print(1.234)
print(1.2e3) # 1200.0

print(0B11.01)  # SyntaxError: invalid syntax
```

2.7 Complex Data Type

A complex number is of the form `a + bj` , a and b contain intergers or floating point values

- In the `real part` (a) –> can use decimal, octal,binary or hexa decimal form.
- But `imaginary part` (bj) –> can use only decimal form.

Note

- Complex data type has some inbuilt attributes to retrieve the real part and imaginary part

```
c = 10.5+3.6j
c.real  # 10.5
c.imag  # 3.6
```

2.8 bool data type

- We can use this data type to represent `boolean values`
- The only allowed values for this data type are - `True` and `False`
- Internally Python represents `True as 1` and `False as 0`

```
bool(0)  # False
# All other no is True

bool('')  # False
# All other string is True


True + True  # 2
True + False # 1
True - True  # 0
```

2.9 str type

A String is a `sequence of characters` enclosed within single quotes or double quotes or triple quotes.

```
a1 = 'Amrit'

a2 = '''Amrit
        Prasad'''

a3 = '''This "Python class very helpful" for java students'''
```

Slicing of Strings

- Slice means a piece
- `[ ] operator` is called slice operator, which can be used to retrieve parts of String.

```
a1 = 'Amrit'

-5 -4 -3 -2 -1
 A  m  r  i  t
 0  1  2  3  4

a1[2:]  # rit
a1[:2]  # Am

a1[::2]  # 'Art'

#  a1[::-1]  # 'tirmA'
```

2.10 bytes Data type

Bytes data type represens a group of byte numbers just like an array.

The only allowed values for byte data type are 0 to 256. By mistake if we are trying to provide any other values then we will get value error.

Once we creates bytes data type value, we cannot change its values,otherwise we will get TypeError.

```
x = [10,20,30,40]
b = bytes(x)

type(b)  # bytes
print(b[0])   # 10
print(b[-1])  # 40

# TypeError: 'bytes' object does not support item assignment
b[0]=100

# ValueError: byte must be in range(0, 256)
bytes([277])
```

2.11 bytearray Data type

Bytearray is exactly same as bytes data type except that its elements can be modified

```
x = [10,20,30,40]
b = bytearray(x)

type(b)  # bytes
print(b[0])   # 10
print(b[-1])  # 40

# No Error
b[0]=100

# ValueError: byte must be in range(0, 256)
bytearray([277])
```

2.12 frozenset Data type

It is exactly same as set except that it is `immutable`. Hence we cannot use add or remove functions.

```
s={10,20,30,40}
type(s)  # set

fs = frozenset(s)
type(fs)  # frozenset
```

2.13 range Data Type

- range Data Type represents a `sequence of numbers`.
- The elements present in range Data type are not modifiable. i.e range Data type is immutable.

```
range(10) # range(0, 10)
generate numbers from 0 to 9

range(10,20)
# generate numbers from 10 to 19

r = range(10,20,2)
# 2 means increment value
for i in r : print(i)   # 10,12,14,16,18
```

2.14 Python collections

- Python has `4 built-in` data structures that can be used to hold a collection of objects,
- They are `list` , `tuple` , `set` , and `dictionary`

[2.14.1 list]

```
# Def empty list
p = list()
p = []

nlist = [10, 10.5, 'ap', True, 20]
```

[2.14.2 Dictionary]

```
# Def empty dict
p = dict()
p = {}

dict([(1,2), (3,4)])
# {1: 2, 3: 4}

{'a': 1, 3: 'aa'}
```

2.14.3 set

```
# Def empty set
p = set()
# p = {}   # `Not set but dict`

{1,2,'a'}

# Fxn
* add()
* update()
* remove()

# Element contains in set must be `immutable type only`
p.add([1,2])  # --> Error
p.add((1,2))  # --> Ok
```

2.14.4 tuple

```
# Def empty tuple
p = ()
p = tuple()

p = 1,2,3,4
# (1, 2, 3, 4)

# Convert into tuples
tuple([1,2,3])
# (1,2,3)

tuple("Amrit")
# ('A', 'm', 'r', 'i', 't')
```

2.15 Data Types Differences

```
# Mutable objects
* bytearray
* list, dict, set
* dict --> maps immutable keys to mutable values
```

```
# Immutable objects
* int, float, complex, bool, str
* bytes, frozenset, range
* tuple
```

```
# Unordered
* dict, set
```

```
# Homogeneous
* str
```

2.16 Hashable

- `Immutable types are hashable`
- But mutable types (lists, dictionaries, set) are not.

.

.

.

.

.

.

.

.

.

.