Python 101

Data Structures in Python (Part-1) - Data Types

Contents

Numbers(int, float, complex, Fractions), their properties and methods

Int and float both are objects in python just like anything else so they have their properities.

We can look at their properties using dir function. (ignore __x__ methods)

int and float both do not have any special properties. Let us explore complex and Rational numbers just a bit.

Although not necessary but in a few programming exercises it is useful to know about complex and Fractions.

Try out the following code on the interpreter

```
>>>  complex1 = 2+3j
>>> complex2 = 4+5j
>>> complex1.imag
3
>>> complex1.real
2
>>> complex1 + complex2 ← performs complex addition
6+8j
>>> 1+2j.imag ← wrong statement if we write it like this Python thinks that we mean (1+2)j
3.0
>>> (1+2j).imag ← this is a proper way
2.0
>>> import fractions
>>> fraction1 = fractions.Fraction(1,2) ← 1/2
>>> fraction2 = fractions.Fraction(2,9) ← 2/9
>>> fraction1.numerator
1
>>> fraction1.denominator
2
>>> fraction1 + fraction2
Fraction(13, 18)
```

Special Numbers in Python

There are 2 special numbers in python,

inf and nan

-inf is less than every number +inf is greater than every number and nan is anything that is not a number.

All of these have internal applications in mathematical modules used in machine learning eg. Numpy, Scipy. So it is useful to know that they exist in python.

Try out the following code on the interpreter

```
>>>  import math
>>> math.inf
inf
>>> - math.inf
-inf
>>> math.nan
nan
```

String, their properties and methods

We already know that strings are a sequence of characters. And they can be represented using double or single quotes.

Strings are immutable in python.

But they are iterable just like lists.

So we can access a character in a string like we access an element in a list.

eg. x = “abcd”

x[0] ← it will print a

String Methods

Type Check Methods -

These set of methods check whether a string is string representation of something in python. eg. String representation of int, alphabet only, numeric only, alphanumeric etc.

Important type check methods are-

1. isdigit() ← check if integer
2. isalnum() ← check if alphanumeric
3. isalpha() ← check if alphabetic
4. isnumeric() ← check if numeric value
5. islower() ←- check if only lowercase
6. isupper() ← -check if only uppercase
7. isspace() ← checks if whitespace only

Case Change Methods -

1. upper() ← return a new string that has all lower case converted to uppercase
2. lower() ← return a new string that has all upper case to lower case
3. swapcase() ← return a new string that has all the cases swapped

String Methods

split, strip and join -

1. split(sub_string) ← returns a list of elements after splitting the string at sub string
   >>> string = “p1y1t1h1o1n1”
   >>> sub_string_to_split_at = ‘1’
   >>> string.split(sub_string_to_split_at)
   >>> [‘p’, ’y’, ’t’, ’h’, ’o’ ,’n’]
2. strip(sub_string) ← returns a new string with the start and end stripped off by the given sub string
   >>> string=”abcPythonabc”
   >>> sub_string_to_strip = ‘abc’
   >>> string.strip(sub_string_to_strip)

   If no sub strings are given in method 1 and 2 then it uses white space. Ie. it will strip white space from start and end and split string from whitespace.
3. join(iterable_of_strings) ← returns a new string which has all of the elements of iterable joined with the string.
   >>> string = “_”
   >>> iterable_of_strings = [‘string1’, ’string2’, ‘string3’]
   >>> string.join(iterable_of_strings)
   string1_string2_string3

String Methods

Other methods-

1. count(sub_string) ← count the occurence of a sub string in a string
2. endswith(sub_string) ← Check if string ends with the specified sub_string
3. startswith(sub_string) ← check if string startswith sub_string
4. replace(sub_str_to_replace, string_to_replace_with, number_of_times_to_replace) ← returns a new string which has all the sub_str_to_replace , replaced with string_to_replace_with for number_of_times_to_replace.
   >>> “PabcdYabcdTabcdHabcdOabcdNabcd”.replace(“abcd”, “”)
   PYTHON

   Mind that if no number_of_times_to_replace are given then it will replace all of the occurences.

We need not remember any of the string methods.
We can check it at the time of coding simply by typing in interpreter-

>>> dir(str)

Then for a specific method’s help we can do
>>> help(**str.**method_name)

String with Flags

Flags convey information about the type of a string, a string can be raw string, formatted string, unicode string or a byte string.

1. **raw strings -** Strings with no evaluation of escape sequences.
   >>> print(r'\n')
   \n
2. **Unicode strings -** Unicode as we know is a character encoding that comprises of all the characters used in all of the languages. The strings which have all their characters in unicode start with flag u. But in python3+ all strings are by default unicoded.
   >>> print(u'Guido Von Rossum')
3. **Byte Strings -** An string may be used to represent a byte sequence in python as well.
   >>>type(b'123')
   <class 'bytes'>
4. **Format Strings -** These strings format on composition by evaluating the expression in paranthesis.
   >>> x = 1
   >>> y = 2
   >>> f'value of x is {x} and for y it is {y} and sum is {x+y}'
   value of x is 1 and for y it is 2

String encoding and decoding

For characters conversion from bytes to letters was called encoding. Similarily for String encoding is applying character encoding at each character in a string.

Now as encoding is conversion from bytes to letters so it is obvious that decoding is conversion from letters to bytes. We can encode and decode strings like following -

`>>> "something".encode()` ← no argument to encode means encoding in utf-8 otherwise valid arguments are 'ascii', 'utf-16' etc.

`>>> b"something".decode()` ← Here also the argument is character encoding.

Indexing and Slicing in String

Indexing means accessing value at a particular index. Which we are already familiar with.

Slicing means accessing value from one index to other index. We can do it using slice operator **[start:end: step]**

```
>>> string = “012345”

>>> string[1:5:2]

“13”
```

We can also reverse string using [::-1] .

Binary, Hex, Bytes and ByteArray

Binary is represented as 0b10101… in Python. Its type is still int but the base on which its working is 2.

eg. 0b01 + 0b10 = 3

We can convert any integer to binary using bin function.

```
>>> bin(3)

'0b11'
```

Mind that the return value of bin function is not integer but string.

Binary, Bytes and ByteArray contd..

We know that Bytes are represented using b flag with strings. These are a different literal. But as they are string with b flag thus they have all the properties of a String. Being String they are **immutable.**

We can also represent hexadecimals using \x followed by valid hexadecimal in a byte string. eg. b'\x01\x0a\x0b'

Bytearrays are essentially an array of bytes. They might look like normal bytes at a glance but we can perform **pop and insert** operations on them which in case of .bytes we can not. Byte arrays are **mutable.**

Type Conversions

Conversion to Bytes-

Only string and integers can be converted to bytes. In order to convert strings to bytes we must also provide the encoding the string is in (eg. utf-8).

```
>>> bytes("string", "utf-8")

>>> bytes(1)
```

Conversion to Bytearray-

It follows same rules as of bytes the only difference is that instead of bytes we called **bytearray**

Type Conversions

Conversion to Boolean-

```
>>> bool(“”) = False

>>> bool(“asd”) = True

>>> bool(0) = False

>>> bool(1.2) = True
```

Except “” empty string and 9 every other value produces True in boolean conversion.

Type Conversions

Conversion to Integer-

```
>>> int(“123”) = 123
```

It also takes an optional parameter that is base. Which is used for interpreting different base values.

eg. conversion of 100 with base 2.

```
>>> int(“100”, 2) = 4
```

Type Conversions

Conversion to Integer-

```
>>> int(“123”) = 123
```

It also takes an optional parameter that is base. Which is used for interpreting different base values.

eg. conversion of 100 with base 2.

```
>>> int(“100”, 2) = 4
```

Conversion to float- `>>> float(“1.1”) = 1.1`

Type Conversions

Conversion to string-

```
>>>str(any_object) = object's_string_representation
```

This is true for all objects ie. everything in python.

The reason behind it is that every object has a dunder method **__str__** which returns its string representation.

Interview Questions

What is the difference between byte and bytearray?

Bytearray is mutable and byte is immutable.

What is format string?

It is a string with capability of evaluating the expressions between the paranthesis.

*It is also called **string interpolation or templating** in languages like JS ES6.*

How can we convert any object to string but its not true for int, float, byte etc? And are objects actually converted to string?

Every object in Python internally implements ___str__ function which returns the string representation of that function. So when we try **str(any_object)** the __str__ method of object is called and it returns the string representation of that object.

No objects are not actually converted to string. They only return their equivalent string representations.

Give a practical usecase of bytearrays?

These can be used for byte buffering.

How would you implement Fractional numbers in Python?

These can be implemented using **fractions** module as following.

```
>>> import fractions

>>> f1 = fractions.Fraction(1,2)

>>> f2 = fractions.Fraction(2,4)

>>> f1+f2

Fraction(1,1)
```

What is the type of inf, and nan? Give the basics of inf and nan arithmetic?

These are floats. But do not behave as typical floats. Their arithmetic rules are as follows-

inf *-1 = -inf	inf+inf = inf
inf-inf = nan	inf * x = inf
nan* x = nan	nan(+,-,/, *) nan = nan