Basic Python by examples

Python interactive: using Python as a calculator

Start Python (or IDLE, the Python IDE).
A prompt is showing up:

```
>>>
```

Display version:

```
>>>help()
```

Help commands:

modules:	available modules
keywords:	list of reserved Python keywords
quit:	leave help

To get help on a keyword, just enter it's name in help.

Simple calculations in Python

```
>>> 3.14*5
15.700000000000001
```

Supported operators:

Operator		Example	Explication
+, - *, /	add, substract, multiply, divide		
%	modulo	25 % 5 = 0 84 % 5 = 4	25/5 = 5, remainder = 0 84/5 = 16, remainder = 4
**	exponent	2**10 = 1024	
// floor division 84//5 = 16 84/5 = 16, remainder = 4	`import division`		

Builtin functions:

```
>>> hex(1024)
'0x400'

>>> bin(1024)
'0b10000000000'
```

Expressions:

```
>>> (20.0+4)/6
4
>>> (2+3)*5
25
```

Using variables

To simplify calculations, values can be stored in variables, and and these can be used as in normal mathematics.

```
>>> a=2.0
>>> b = 3.36
>>> a+b
5.359999999999999
>>> a-b
-1.3599999999999999
>>> a**2 + b**2
15.289599999999998
>>> a>b
False
```

The name of a variable must not be a Python keyword!

Python Functions
Functions are the most important aspect of an application. A function can be defined as the organized block of reusable code which can be called whenever required.

Python allows us to divide a large program into the basic building blocks known as function. The function contains the set of programming statements enclosed by {}. A function can be called multiple times to provide reusability and modularity to the python program.

In other words, we can say that the collection of functions creates a program. The function is also known as procedure or subroutine in other programming languages.

Advantage of Functions in Python

There are the following advantages of Python functions.

- By using functions, we can avoid rewriting same logic/code again and again in a program.
- We can call python functions any number of times in a program and from any place in a program.
- We can track a large python program easily when it is divided into multiple functions.
- Reusability is the main achievement of python functions.
- However, Function calling is always overhead in a python program.

Creating a function

In python, we can use **def** keyword to define the function. The syntax to define a function in python is given below.

```
def my_function():
    function-suite
    return <expression>
```

The function block is started with the colon (:) and all the same level block statements remain at the same indentation.

A function can accept any number of parameters that must be the same in the definition and function calling.

Function calling

In python, a function must be defined before the function calling otherwise the python interpreter gives an error. Once the function is defined, we can call it from another function or the python prompt. To call the function, use the function name followed by the parentheses.

A simple function that prints the message "Hello Word" is given below.

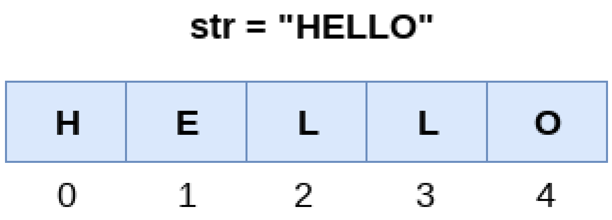
```python
def hello_world():
    print("hello world")

hello_world()
```

Output:

```
hello world
```

Strings indexing and splitting

Like other languages, the indexing of the python strings starts from 0. For example, The string "HELLO" is indexed as given in the below figure.



str[0] = 'H'

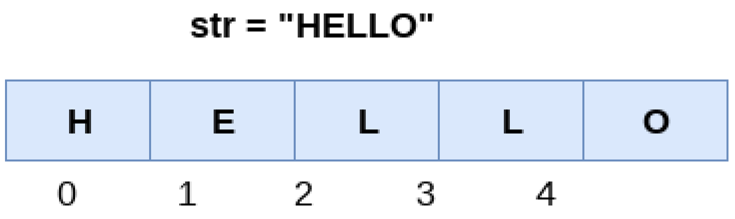
str[1] = 'E'

str[2] = 'L'

str[3] = 'L'

str[4] = 'O'

As shown in python, the slice operator [] is used to access the individual characters of the string. However, we can use the : (colon) operator in python to access the substring. Consider the following example.



```
str[0] = 'H'        str[:] = 'HELLO'

str[1] = 'E'        str[0:] = 'HELLO'

str[2] = 'L'        str[:5] = 'HELLO'

str[3] = 'L'        str[:3] = 'HEL'

str[4] = 'O'        str[0:2] = 'HE'

                    str[1:4] = 'ELL'
```

Built-in String functions

Python provides various in-built functions that are used for string handling. Many String fun

Method	Description
capitalize()	It capitalizes the first character of the String. This function is deprecated in python3
casefold()	It returns a version of s suitable for case-less comparisons.
center(width ,fillchar)	It returns a space padded string with the original string centred with equal number of left and right spaces.
count(string,begin,end)	It counts the number of occurrences of a substring in a String between begin and end index.
decode(encoding = 'UTF8', errors = 'strict')	Decodes the string using codec registered for encoding.
encode()	Encode S using the codec registered for encoding. Default encoding is 'utf-8'.
endswith(suffix ,begin=0,end=len(string))	It returns a Boolean value if the string terminates with given suffix between begin and end.
expandtabs(tabsize = 8)	It defines tabs in string to multiple spaces. The default space value is 8.
find(substring ,beginIndex, endIndex)	It returns the index value of the string where substring is found between begin index and end index.
format(value)	It returns a formatted version of S, using the passed value.
index(subsring, beginIndex, endIndex)	It throws an exception if string is not found. It works same as find() method.
isalnum()	It returns true if the characters in the string are alphanumeric i.e., alphabets or numbers and there is at least 1 character. Otherwise, it returns false.
isalpha()	It returns true if all the characters are alphabets and there is at least one character, otherwise False.
isdecimal()	It returns true if all the characters of the string are decimals.
isdigit()	It returns true if all the characters are digits and there is at least one character, otherwise False.
isidentifier()	It returns true if the string is the valid identifier.
islower()	It returns true if the characters of a string are in lower case, otherwise false.

isnumeric()	It returns true if the string contains only numeric characters.
isprintable()	It returns true if all the characters of s are printable or s is empty, false otherwise.
isupper()	It returns false if characters of a string are in Upper case, otherwise False.
isspace()	It returns true if the characters of a string are white-space, otherwise false.
istitle()	It returns true if the string is titled properly and false otherwise. A title string is the one in which the first character is upper-case whereas the other characters are lower-case.
isupper()	It returns true if all the characters of the string(if exists) is true otherwise it returns false.
join(seq)	It merges the strings representation of the given sequence.
len(string)	It returns the length of a string.
ljust(width[,fillchar])	It returns the space padded strings with the original string left justified to the given width.
lower()	It converts all the characters of a string to Lower case.
lstrip()	It removes all leading whitespaces of a string and can also be used to remove particular character from leading.
partition()	It searches for the separator sep in S, and returns the part before it, the separator itself, and the part after it. If the separator is not found, return S and two empty strings.
maketrans()	It returns a translation table to be used in translate function.
replace(old,new[,count])	It replaces the old sequence of characters with the new sequence. The max characters are replaced if max is given.
rfind(str,beg=0,end=len(str))	It is similar to find but it traverses the string in backward direction.
rindex(str,beg=0,end=len(str))	It is same as index but it traverses the string in backward direction.
rjust(width,[,fillchar])	Returns a space padded string having original string right justified to the number of characters specified.

rstrip()	It removes all trailing whitespace of a string and can also be used to remove particular character from trailing.
rsplit(sep=None, maxsplit = -1)	It is same as split() but it processes the string from the backward direction. It returns the list of words in the string. If Separator is not specified then the string splits according to the white-space.
split(str,num=string.count(str))	Splits the string according to the delimiter str. The string splits according to the space if the delimiter is not provided. It returns the list of substring concatenated with the delimiter.
splitlines(num=string.count('\n'))	It returns the list of strings at each line with newline removed.
startswith(str,beg=0,end=len(str))	It returns a Boolean value if the string starts with given str between begin and end.
strip([chars])	It is used to perform lstrip() and rstrip() on the string.
swapcase()	It inverts case of all characters in a string.
title()	It is used to convert the string into the title-case i.e., The string **meEruT** will be converted to Meerut.
translate(table,deletechars = '')	It translates the string according to the translation table passed in the function .
upper()	It converts all the characters of a string to Upper Case.
zfill(width)	Returns original string leftpadded with zeros to a total of width characters; intended for numbers, zfill() retains any sign given (less one zero).

1. List indexing and splitting

The indexing are processed in the same way as it happens with the strings. The elements of the list can be accessed by using the slice operator [].

The index starts from 0 and goes to length - 1. The first element of the list is stored at the 0th index, the second element of the list is stored at the 1st index, and so on.

Consider the following example.

List = [0, 1, 2, 3, 4, 5]

0	**1**	**2**	**3**	**4**	**5**

List[0] = 0

List[1] = 1

List[2] = 2

List[3] = 3

List[4] = 4

List[5] = 5

List[0:] = [0,1,2,3,4,5]

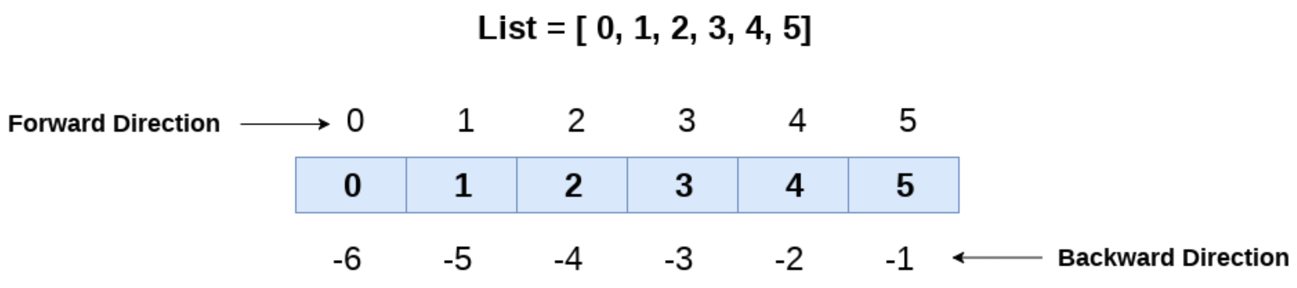
List[:] = [0,1,2,3,4,5]

List[2:4] = [2, 3]

List[1:3] = [1, 2]

List[:4] = [0, 1, 2, 3]

Unlike other languages, python provides us the flexibility to use the negative indexing also. The negative indices are counted from the right. The last element (right most) of the list has the index -1, its adjacent left element is present at the index -2 and so on until the left most element is encountered.



2. Updating List values

Lists are the most versatile data structures in python since they are immutable and their values can be updated by using the slice and assignment operator.

Python also provide us the append() method which can be used to add values to the string.

Consider the following example to update the values inside the list.

```
List = [1, 2, 3, 4, 5, 6]
print(List)
List[2] = 10;
print(List)
List[1:3] = [89, 78]
print(List)
```

Output:

```
[1, 2, 3, 4, 5, 6]
[1, 2, 10, 4, 5, 6]
[1, 89, 78, 4, 5, 6]
```

Python List built-in methods

SN	Function	Description
1	list.append(obj)	The element represented by the object obj is added to the list.
2	list.clear()	It removes all the elements from the list.
3	List.copy()	It returns a shallow copy of the list.
4	list.count(obj)	It returns the number of occurrences of the specified object in the list.
5	list.extend(seq)	The sequence represented by the object seq is extended to the list.
6	list.index(obj)	It returns the lowest index in the list that object appears.
7	list.insert(index, obj)	The object is inserted into the list at the specified index.
8	list.pop(obj=list[-1])	It removes and returns the last object of the list.
9	list.remove(obj)	It removes the specified object from the list.
10	list.reverse()	It reverses the list.
11	list.sort([func])	It sorts the list by using the specified compare function if given.