

RGUKT-IIIT Srikakulam, Dept. of CSE

Python unit 3 tutorials

Python Variables

Variable is a name which is used to refer memory location. Variable also known as identifier and used to hold value.

- A variable name must start with a letter or the underscore character.
- A variable name cannot start with a number.
- The equal (=) operator is used to assign value to a variable.
- Non technically, you can suppose variable as a bag to store books in it and those books can be replaced at any time.

Note - Variable name should not be a keyword.

Declaring Variables in Python

```
counter = 100      # An integer assignment
miles  = 1000.0    # A floating point
name   = "John"    # A string
print counter
print miles
print name
```

2. Assigning multiple values to multiple variables:

Example:

1. a,b,c=5,10,15
2. `print a`
3. `print b`
4. `print c`

Python Keywords

- Python Keywords are special reserved words which convey a special meaning to the compiler/interpreter.
- Each keyword have a special meaning and a specific operation.
- These keywords can't be used as variable.
- Following is the List of Python Keywords.

True	False	None	and	as
assert	def	class	continue	break
Else	finally	elif	del	except
global	for	if	from	import
raise	try	or	return	pass
nonlocal	in	not	is	lambda

Python Data Types

Data Types in Python(Python Collections)

Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

There are various data types in Python. Some of the important types are listed below.

- [Data Types in Python](#)
 - [Python Numbers](#)
 - [Python List](#)
 - [Python Tuple](#)
 - [Python Strings](#)
 - [Python Set](#)
 - [Python Dictionary](#)

Python Numbers

Integers, floating point numbers and complex numbers falls under [Python numbers](#) category. They are defined as `int`, `float` and `complex` class in Python.

There are three numeric types in Python:

- `int`
- `float`
- `complex`

Example

```
x = 1 # int
y = 2.8 # float
z = 1j # comple
```

To verify the type of any object in Python, use the `type()` function:

Example

```
print(type(x))
print(type(y))
print(type(z))
```

Output:

```
<class 'int'>
```

```
<class 'float'>
```

```
<class 'complex'>
```

Int:

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

Example

Integers:

```
x = 1
```

```
y = 35656222554887711
```

```
z = -3255522
```

```
print(type(x))
```

```
print(type(y))
```

```
print(type(z))
```

output:

```
<class 'int'>
```

```
<class 'int'>
```

```
<class 'int'>
```

Float:

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

Example

Floats:

```
x = 1.10
```

```
y = 1.0
```

```
z = -35.59
```

```
print(type(x))
```

```
print(type(y))
```

```
print(type(z))
```

output:

```
<class 'float'>
```

```
<class 'float'>
```

```
<class 'float'>
```

Complex

Complex numbers are written with a "j" as the imaginary part:

Example

Complex:

x = 3+5j

y = 5j

z = -5j

```
print(type(x))
```

```
print(type(y))
```

```
print(type(z))
```

output:

```
<class 'complex'>
<class 'complex'>
<class 'complex'>
```

Boolean:

A **boolean** expression (or logical expression) evaluates to one of two states true or false. **Python** provides the **boolean** type that can be either set to False or True. Many functions and **operations** returns **boolean** objects.

```
>>> 2+2==4
```

```
True
```

```
>>> 4-3==5
```

```
False
```

```
>>> None
```

```
>>> True
```

```
True
```

```
>>> False
```

```
False
```

Note: Do more examples for **In** and **Not** in operations among list, tuple, string, set and Dictionaries.

None:

Think of a scenario where you don't want the variable to have any value. What will do? You can't use 0 because it's a number after all. That is where **None** comes into picture. Just put:

```
>>> x = None
```

And **x** will have no value.

Convert Decimal Number into Binary, Octal and Hexadecimal Number System

- ✓ Decimal system is the most widely used number system.
- ✓ But computer only understands binary.
- ✓ Binary, octal and hexadecimal number systems are closely related and we may require to convert decimal into these systems.
- ✓ Decimal system is base 10 (ten symbols, 0-9, are used to represent a number) and similarly, binary is base 2, octal is base 8 and hexadecimal is base 16.
 - A number with the prefix '0b' is considered binary,
 - '0o' is considered octal and
 - '0x' as hexadecimal.

Example:

```
>>> bin(8)
'0b1000'
>>> oct(8)
'0o10'
>>> hex(16)
'0x10'
```

Python List

- List is an ordered sequence of items.
- A list is a collection which is ordered and changeable.
- It is one of the most used datatype in Python and is very flexible.
- All the items in a list do not need to be of the same type.
- A list can be composed by storing a sequence of different type of values separated by commas.
- Python list is enclosed between square [] brackets and elements are stored in the index basis with starting index 0.

Example:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

Output:

```
['apple', 'banana', 'cherry']
```

	Create >>>List=[2,4,"grapes"]	Create >>>List2=[2,5,3,8,4]	
	List Operation	Command	Description
1	Replace	>>>list1[2]="mango"	(2,4,"mango")
2	insert	>>>list.insert(1,"orange")	(2,"orange",4,"mango")
3	sort	>>>list2.sort()	(2,3,4,5,8)

4	reverse	>>>list2.reverse()	(8,5,4,3,2)
5	append	>>>list2.append("guava")	(8,5,4,3,2,"guava")
6	delete	>>>del list2[5]	(8,5,4,3,2)
7	concatenation	>>>list1+list2	(2,"orange",4,"mango",8,5,4,3,2)
8	nested list example ; >>>list3=[(2,1),(4,8),(5,6)]		
9	delete	>>>del list3[2]	[(2,1),(4,8)]
10	Pop	>>>list2.pop()	(8,5,4,3,2)
11	Pop with index	>>>list2.pop(1)	(8,4,3,2)

Characteristics of List.

- 1.Mutable
- 2.Linear Data structure
- 3.Mixed type Elements
- 4.Variable Length
- 5.Zero Based Indexing

Python Tuple

A tuple is a sequence of immutable objects, therefore tuple cannot be changed.

- It can be used to collect different types of object.
- The objects are enclosed within parenthesis and separated by comma.
- Tuple is similar to list.
- Only the difference is that list is enclosed between square bracket, tuple between parenthesis and List has mutable objects whereas Tuple has immutable objects.

Tuple Example.

```
>>> thistuple = ("apple", "banana", "cherry")
```

```
>>> print(thistuple)
```

```
('apple', 'banana', 'cherry')
```

```
>>> data=(10,20,'ram',56.8)
```

```
>>> data2="a",10,20.9
```

```
>>> data
(10, 20, 'ram', 56.8)
>>> data2
('a', 10, 20.9)
```

Python Empty Tuple Example

```
tuple1=()
>>> tuple=()
>>> tuple
```

Python Single Object Tuple Example

For a single valued tuple, there must be a comma at the end of the value.

```
>>> tuple=(10,)
>>> tuple
(10,)
```

Python Tuple of Tuples Example

- Tuples can also be nested; it means we can pass tuple as an element to create a new tuple.
- See, the following example in which we have created a tuple that contains tuples an the object.

Example;

```
>>> tupl1=('a','mahesh',10.56)
>>> tupl2=tupl1,(10,20,30)
>>> tupl1
('a', 'mahesh', 10.56)
>>> tupl2
(('a', 'mahesh', 10.56), (10, 20, 30))
```

PYTHON STRINGS

- Python string is a built-in type text sequence.
- It is used to handle textual data in python.
- Creating Strings are simplest and easy to use in Python.
- We can simply create Python String by enclosing a text in single as well as double quotes.
- Python treat both single and double quotes statements same.

Create >>>Str1="hello" >>>Str2="world"

string operation	Command	Description
1.concatenation	>>>str1+str2	"helloworld"
2.repetition	>>>str1*2	'hellohello'
#repeating the string		
3.slice	>>>str1[1]	'e'
#str1="hello" print the 2 nd character of hello ie...here [h refers to 0 and 1 st character; e refers to 1 and 2 nd character]		
4.range slice	>>>str2[2:]	'rld'
#str2[2:] means print the characters of string except first three		
5.range slice	>>>str2[0:1]	'w'
6.membership[in]	>>>'g' in str1	True
7.membership[not in]	>>>'h' not in str2	False

PYTHON SEQUENCES (LIST, TUPLE, STRING)

>>>List1=[2,4,1,6] >>>Tuple=(5,6,8,3) >>>Str="python"
>>>List2=[5,5,4,3,2] >>>Str2="banana" >>>A={2,3,4,5,6}
>>>B={8,4,3,5,'grapes'}

Sequence OPERATION	COMMAND	DESCRIPTION
1.Length	>>>len(list1)	4
	>>>len(tuple)	4
	>>>len(str)	6
2.Select	>>>list1[1]	4
	>>>tuple[2]	8
	>>>str[0]	p
3.Slice	>>>list1[1:3]	(4,1)
	>>>tuple[2:4]	(8,3)
	>>>str[0:6]	'python'
4.count	>>>list1.count(2)	1
	>>>tuple.count(3)	1
	>>>str.count('t')	1
	>>>list2.count(5)	2
	>>>str2.count('a')	3
#count means how many times the given element is repeating in the given list or tuple or string will be printed		
5.Index	>>>list1.index(6)	3
	>>>tuple.index(8)	2
	>>>str.index('h')	3
#index will define the position of the given element		
6.membership	>>>4 in list1	True
	>>>s not in tuple	False
	>>>'y' in str	True
#membership is defined by Boolean operation[in, not in which will be defined as true or false]		
7.Concatenation	>>>list1+list2	-----
	>>>tuple+tuple2	-----
	>>>str+str2	-----
8.Minimum value	>>>min(list1)	1
	>>>min(tuple)	3
	>>>min(str)	h

9.Maximum value	>>>max(list1)	6
	>>>max(tuple)	8
	>>>max(str)	y
10.Sum	>>>sum(list1)	13
	>>>sum(tuple)	22
#Sum not possible in string because sum is not applicable for alphabets		

Python Set

- set is an unordered list
- set is a mutable data type like list
- to create set use curly braces '{ }'

Example1 >>>fruits={'apple','banana','mango'}

 >>>fruits

 {'apple',' banana','mango'}

Example2 >>>num={2,4,5,6,2,8,5}

>>>A={2,3,4,5,6} >>>B={8,4,3,5,'grapes'}

SET OPERATION	COMMAND	DESCRIPTION
1.Add	>>>A.add(8)	{2,3,4,5,6,8}
2.Remove	>>>A.remove(5)	{2,3,4,6,8}
3.Union	>>>A B	{2,3,4,5,6,8,'grapes'}
4.Intersection	>>>A&B	{8,3,4}
5.Difference	>>>A-B	{2,6}
	>>>B-A	{5,'grapes'}
6.copy	>>>Z=A.copy()	{2,3,4,6,8}
7.size	>>>len(A)	5
8.symmetric difference	>>>A^B	{2,5,6,'grapes'}
9.membership	>>>5 in A	False
	>>>'grapes' in B	True
10.Clear	>>>A.clear	
	>>>A	set()

Python Dictionaries

- The python dictionary is called as associative data structure.
- Python dictionary is a mutable data type.
- To create dictionary use curly braces '{ }' in between use (,) for separate items and use (:)colon for key and its value.
- Dictionary will have keys and its values
- There are three ways to create python dictionaries,

Example 1:

>>>Num={1:"one",2:'two',3:'three'}

Here '1' is called as key and 'one' is called as value

Example 2: >>>temp= {}

>>>temp['sun']=20

```
>>>temp['mon']=22
>>>temp['tue']=26
>>>temp['wed']=28
>>>temp
{'sun':20,'mon':22,'tue':26,'wed':28}
```

Example:3

➤ Use dict keyword to create empty dictionary

➤ >>>dict={}

For Dictionary Operations use running notes.

Mutable and Immutable Variables:

Mutable vs. Immutable Objects

“Not all objects are created equal”

There are two kinds of objects in Python:

➤ Mutable objects and

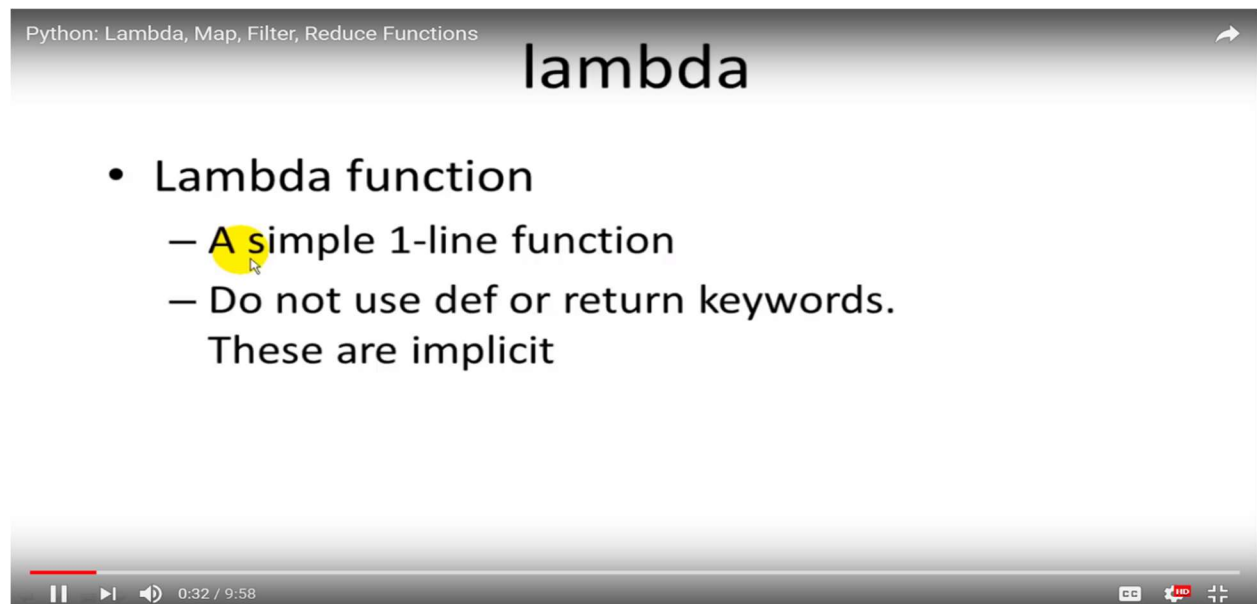
➤ Immutable objects.

✓ The value of a mutable object can be modified ***in place*** after it's creation, while the value of an immutable object cannot be changed.

- **Immutable Object:** *int, float, complex, string, tuple, bool.*
- **Mutable Object:** *list, dict, set.*

Mapping:

Lambda function, Map function, Filter function, Reduce function



The screenshot shows a video player interface. At the top, there is a title bar that reads "Python: Lambda, Map, Filter, Reduce Functions" and a "lambda" heading. Below the heading, there is a list of bullet points: "• Lambda function", "– A simple 1-line function", and "– Do not use def or return keywords. These are implicit". The video player controls at the bottom show a progress bar at 0:32 / 9:58, a play button, and a volume icon.

Python: Lambda, Map, Filter, Reduce Functions

lambda

- Lambda function
 - A simple 1-line function
 - Do not use def or return keywords. These are implicit

0:32 / 9:58

lambda

```
# double x
```

```
def double (x):  
    return x * 2
```

lambda x: 2 * x

Parameter(s) Return

map

- Apply same function to each element of a sequence
- Return the modified list

List, [m, n, p] Function, f() **map** New list, [f(m), f(n), f(p)]

map

```
# prints [16, 9, 4, 1]
```

```
def square (lst1):  
    lst2 = []  
    for num in lst1:  
        lst2.append(num ** 2)  
    return lst2
```

```
print square([4,3,2,1])
```

n = [4, 3, 2, 1]
print (list(map(lambda x: x**2, n)))

Function List

filter

- Filter items out of a sequence
- Return filtered list



filter

```
# prints [4, 3]
```

```
def over_two(lst1):  
    lst2 = [x for x in lst1 if x>2]  
    return lst2  
  
print over_two([4,3,2,1])
```

```
n = [4, 3, 2, 1]  
print (list(filter(lambda x: x>2, n)))
```

Diagram illustrating the filter function process:

Condition: $x > 2$

List: n

Arrows point from the labels 'Condition' and 'List' to the corresponding parts of the lambda function and list in the code above.

reduce

- Applies same operation to items of a sequence
- Uses result of operation as first param of next operation
- Returns an item, not a list



reduce

prints 24

```
def mult (lst1):  
    prod = lst1[0]  
    for i in range(1, len(lst1)):  
        prod *= lst1[i]  
    return prod  
  
print mult([4,3,2,1])
```

```
n = [4, 3, 2, 1]  
print (reduce(lambda x,y: x*y, n))
```

Function List

```
4 * 3 = 12  
12 * 2 = 24  
24 * 1 = 24
```

Thank you

Prepared by

Ramesh Yajjala, Assistant Professor(c)
Dept. of CSE, IIIT-SKLM, RGUKT-AP