



Python

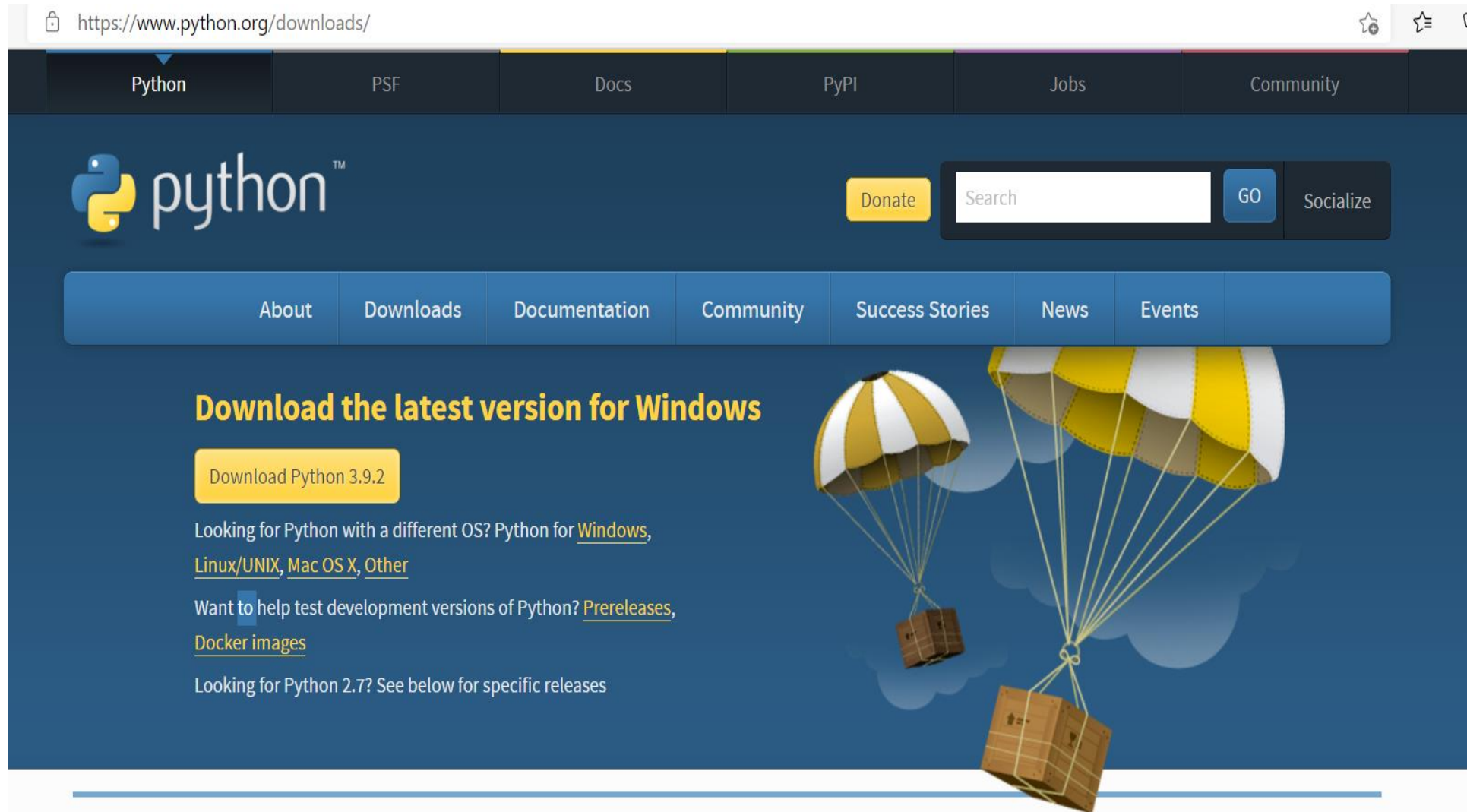


- Programming
- Importing & Cleaning Data
- Data Manipulation
- Data Visualization
- Probability & Statistics
- Machine Learning
- Applied Finance
- Reporting & Case Studies Management
- Customer Analytics and A/B Testing in Python
- Image analysis with Python
- Networking(Juniper and cisco)
- Agriculture
- Automation
- Strong library support



Python

- Guido Van Rossum
- General Purpose: build anything
- Open Source! Free!
- Python Packages, also for Data Science
 - Many applications and fields
- Version 3.x - <https://www.python.org/downloads/>



https://www.youtube.com/watch?v=-nitykYMYsc&list=PLfn1X0acn8m2kLElb6N8m9RPp_WEXbBm6



C:\Windows\system32\cmd.exe

```
Microsoft Windows [Version 10.0.19042.867]  
(c) 2020 Microsoft Corporation. All rights reserved.
```


```
C:\Users\Divya>python --version  
Python 3.8.5
```

```
C:\Users\Divya>_
```



Python releases by version number:

Release version	Release date
Python 3.9.2	Feb. 19, 2021
Python 3.8.8	Feb. 19, 2021
Python 3.6.13	Feb. 15, 2021
Python 3.7.10	Feb. 15, 2021
Python 3.8.7	Dec. 21, 2020
Python 3.9.1	Dec. 7, 2020
Python 3.9.0	Oct. 5, 2020


python™

https://www.anaconda.com/products/individual

Windows

Python 3.8

64-Bit Graphical Installer (457 MB)

32-Bit Graphical Installer (403 MB)

MacOS

Python 3.8

64-Bit Graphical Installer (435 MB)

64-Bit Command Line Installer (428 MB)

Linux

Python 3.8

64-Bit (x86) Installer (529 MB)

64-Bit (Power8 and Power9) Installer (279 MB)

Anaconda IDE





Divya\untitled107.py

untitled102.py × untitled103.py × untitled104.py × untitled105.py × panda_bachelor_degree_ex.py × untitled106.py × untitled107.py ×

Debug file (Ctrl+F5)

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Fri Apr  2 10:14:06 2021
4
5 @author: Divya
6 """
7
8 a=10
9 b=20
10 print(a+b)
11 c=10
12 d=20
13 print(c-d)
14
```

Name	Type	Size	Value
a	int	1	10
b	int	1	20
c	int	1	10
d	int	1	20

Help Variable explorer Plots Files

Console 1/A ×

```
13
In [324]: runfile('C:/Users/Divya/untitled106.py', wdir='C:/Users/Divya')
37.1
13
In [325]: runfile('C:/Users/Divya/untitled107.py', wdir='C:/Users/Divya')
30
-10
In [326]: debugfile('C:/Users/Divya/untitled107.py', wdir='C:/Users/Divya')
> c:\users\divya\untitled107.py(2)<module>()
----> 1 # -*- coding: utf-8 -*-
2 """
3 Created on Fri Apr  2 10:14:06 2021
4
5 @author: Divya
6 """
7
8 a=10
9 b=20
10 print(a+b)
11 c=10
12 d=20
13 print(c-d)
14
```

Itrod

Path to Data science (Data scientist)



- Basics of python
- Programming skills data structures
- Pandas for data manipulation
- Joining data with pandas
- Data manipulation python
- Data visualization Matplotlib /seaborn
- Importing and cleaning the data
- Cleaning of data
- Working date and time in python
- Statistical thinking with python
- Supervised learning with scikit-learn
- Unsupervised learning in python
- ML with tree based Model
- Cluster analysis

Case studies : School budgeting with ML

Analyzing TV data

Visual history of Nobel prize winner



Data Engineer - ETL

- Installation of datawarehouse solution
- Data modelling
- data architect
- Database architect testing

Data Analyst - Data collection and processing

- Programming
- ML
- Data Visualization
- Applying statistical analysis

Data Architect - Dataware housing solution

- Data Modeling (Enterprise)
-

Data Scientist - Data cleaning and processing

- Predictive model
- ML
- Storytelling and Visualization



Python versions

Two major versions of Python are currently in active use:

Python 3.x is the current version and is under active development.

Python 2.x is the legacy version and will receive only security updates until 2020. No new features will be implemented.

Note that many projects still use Python 2, although migrating to Python 3 is getting easier.



Check Current python version

Python --version

 [Python youtube playlist](#)



Different ways to exit python shell

1)*exit()*

2)*quit()*

Alternatively, CTRL + D will close the shell and put you back on your terminal's command line.

If you want to cancel a command you're in the middle of typing and get back to a clean command prompt, while staying inside the Interpreter shell, use CTRL + C

 [Python youtube playlist](#)



Python Script

- Text Files - .py
- List of Python Commands
- Similar to typing in IPython Shell

 [Python youtube playlist](#)



Let's begin our Journey by printing the Hello, World! example.

```
print("Hello, World!")
```

 [Python youtube playlist](#)



Unlike Python2, which did not require you to put a parenthesis, in Python3, parenthesis is a must else it will raise a syntax error as shown below.

```
print "Hello, World!"
```

```
File "<ipython-input-6-a1fcabcd869c>", line 1
```

```
    print "Hello, World!"
```

```
        ^
```

```
SyntaxError: Missing parentheses in call to 'print'. Did you mean print("Hello, World!")?
```

From the above output, you can conclude that in Python3, print() is not a statement but a function.



```
print('datpython', 'tutorial', 'on', 'python', 'print', 'function', sep='\n\n')
```

datpython

tutorial

on

python

print

Function

 [Python youtube playlist](#)



Creating variables and assigning values

To create a variable in Python, all you need to do is specify the variable name, and then assign a value to it.

<variable name> = <value>

Python uses = to assign values to variables. There's no need to declare a variable in advance (or to assign a data type to it), assigning a value to a variable itself declares and initializes the variable with that value

.

There's no way to declare a variable without assigning it an initial value.

 [Python youtube playlist](#)



Creating variables and assigning values cont..

```
# Integer  
a = 2  
print(a)  
# Output: 2
```

```
# Integer  
b = 9223372036854775807  
print(b)  
# Output:  
9223372036854775807
```

```
# Floating point  
pi = 3.14  
print(pi)  
# Output: 3.14
```



Creating variables and assigning values cont..

```
# String  
c = 'A'  
print(c)  
# Output: A
```

```
# String  
name = 'John Doe'  
print(name)  
# Output: John Doe
```

```
# Boolean  
q = True  
print(q)  
# Output: True
```



Rules for variable naming

1. Variables names must start with a letter or an underscore.

```
x = True # valid
_y = True # valid
9x = False # starts with numeral
=> SyntaxError: invalid syntax
$y = False # starts with symbol
=> SyntaxError: invalid syntax
```

2. The remainder of your variable name may consist of letters, numbers and underscores.

```
has_0_in_it = "Still Valid"
```

3. Names are case sensitive.

```
x = 9
y = X*5
=> NameError: name 'X' is not defined
```



Can you answer these:

- 1) When we will get NameError ?
- 2) How to check version?
- 3) Can we declare variable with out data type ?
- 4) When we will get SyntaxError?

Smart Python Interpreter



Even though there's no need to specify a data type when declaring a variable in Python.

while allocating the necessary area in memory for the variable, the Python interpreter automatically picks the most suitable built-in type for it:



```
a = 2
print(type(a))
# Output: <type 'int'>
```

```
b = 9223372036854775807
print(type(b))
# Output: <type 'int'>
```

```
pi = 3.14
print(type(pi))
# Output: <type 'float'>
```

```
c = 'A'
print(type(c))
# Output: <type 'str'>
```

You can assign multiple values to multiple variables in one line.
Note that there must be the same number of arguments on the right and left sides of the = operator:

```
a, b, c = 1, 2, 3  
print(a, b, c)  
# Output: 1 2 3
```

```
a, b, c = 1, 2  
=> Traceback (most recent call last):  
=> File "name.py", line N, in <module>
```

```
=> a, b, c = 1, 2  
=> ValueError: need more than 2 values to unpack
```

```
a, b = 1, 2, 3  
=> Traceback (most recent call last):  
=> File "name.py", line N, in <module>
```

```
=> a, b = 1, 2, 3  
=> ValueError: too many values to unpack
```





Nested lists are also valid in python. This means that a list can contain another list as an element.

```
x = [1, 2, [3, 4, 5], 6, 7] # this is nested list
print x[2]
# Output: [3, 4, 5]
print x[2][1]
# Output: 4
```

Datatypes



Built-in Types

Booleans

bool: A boolean value of either **True** or **False**. Logical operations like **and**, or, **not** can be performed on booleans.

x or y *# if x is False then y otherwise x*
x and y *# if x is False then x otherwise y*
not x *# if x is True then False, otherwise True*

In Python 2.x and in Python 3.x, a boolean is also an **int**. The **bool** type is a subclass of the **int** type and **True** and **False** are its only instances:

```
issubclass(bool, int) # True  
instance(True, bool) # True  
instance(False, bool) # True
```

 [Python youtube playlist](#)

```
# Comparison of Booleans  
print(True == False)
```

```
# Comparison of integers  
print(-5 * 15 != 75)
```

```
# Comparison of strings  
print("pyscript" == "PyScript")
```

```
# Compare a boolean with a numeric  
print(True == 1)
```

Comparison of integers

```
x = -3 * 6
```

```
print(x>=-10)
```

Comparison of strings

```
y = "test"
```

```
print("test"<=y)
```

Comparison of booleans

```
print(True>False)
```



[Python youtube playlist](#)

```
# Define variables  
my_kitchen = 18.0  
your_kitchen = 14.0
```

```
# my_kitchen bigger than 10 and smaller than 18?  
print(my_kitchen >10 and my_kitchen<18)
```

```
# my_kitchen smaller than 14 or bigger than 17?  
print(my_kitchen <14 or my_kitchen>17)
```

```
# Double my_kitchen smaller than triple your_kitchen?  
print(2*my_kitchen < 3*your_kitchen)
```



[Python youtube playlist](#)



Mutable and Immutable Data Types

An object is called *mutable* if it can be changed. For example, when you pass a list to some function, the list can be changed:

```
def f(m):  
    m.append(3) # adds a number to the list. This is a mutation.  
x = [1, 2]  
f(x)  
x == [1, 2] # False now, since an item was added to the list
```

An object is called *immutable* if it cannot be changed in any way. For example, integers are immutable, since there's no way to change them:

```
def bar():  
    x = (1, 2)  
    g(x)  
x == (1, 2) # Will always be True, since no function can change the object (1, 2)
```




Note that **variables** themselves are mutable, so we can reassign the *variable* x, but this does not change the object that x had previously pointed to. It only made x point to a new object.

Data types whose instances are mutable are called *mutable data types*, and similarly for immutable objects and datatypes.



Examples of immutable Data Types:

int, long, float, complex

str

bytes

tuple

Frozenset

Examples of mutable Data Types:

bytearray

list

set

dict

 [Python youtube playlist](#)



Collection Types

There are a number of collection types in Python. While types such as `int` and `str` hold a single value, collection types hold multiple values



List Data Type

A list contains items separated by commas and enclosed within square brackets []. Lists are almost similar to arrays in C. One difference is that all the items belonging to a list can be of different data type.

```
list = [123,'abcd',10.2,'d'] #can be a array of any data type or single data type.
```

```
list1 = ['hello','world']
```

```
print(list) #will output whole list. [123,'abcd',10.2,'d']
```

```
print(list[0:2]) #will output first two element of list. [123,'abcd']
```

```
print(list1 * 2) #will gave list1 two times. ['hello','world','hello','world']
```

```
print(list + list1) #will gave concatenation of both the lists.  
[123,'abcd',10.2,'d','hello','world']
```

 [Python youtube playlist](#)

1. List doesn't support arithmetic operations
2. List supports negative indexing
3. List supports slicing
4. List supports insertion order
5. List supports duplicate elements
6. List is mutable data type
7. List is stored in []

Insert vs append

Remove vs pop

Empty list size is

```
b=[ ]  
print(b.__sizeof__())
```

Exercise -1



area variables (in square meters)

hall = 11.25

kit = 18.0

liv = 20.0

bed = 10.75

Bath = 9.50

- Create a list, areas, that contains the area of the hallway (hall), kitchen (kit), living room (liv), bedroom (bed) and bathroom (bath), in this order.
- Use the predefined variables.
- Print areas with the print() function.



Exercise

```
# area variables (in square meters)
```

```
hall = 11.25
```

```
kit = 18.0
```

```
liv = 20.0
```

```
bed = 10.75
```

```
bath = 9.50
```

```
# Create list areas
```

```
areas=[hall,kit,liv,bed,bath]
```

```
# Print areas
```

```
print(areas)
```


Exercise -2



```
kit = 18.0
```

```
liv = 20.0
```

```
bed = 10.75
```

```
bath = 9.50
```

```
# Adapt list areas
```

```
areas = [hall, kit, "living room", liv, bed, "bathroom", bath]
```

```
# Print areas
```

Finish the line of code that creates the areas list such that the list first contains the name of each room as a string and then its area. More specifically, add the strings "hallway", "kitchen" and "bedroom" at the appropriate locations.

"bathroom" is a string, while bath is a variable that represents the float 9.50

Print areas again; is the printout more informative this time?

Exercise -4



```
# area variables (in square meters)
```

```
hall = 11.25
```

```
kit = 18.0
```

```
liv = 20.0
```

```
bed = 10.75
```

```
bath = 9.50
```

```
# house information as list of lists
```

```
house = [ ["hallway", hall], ["kitchen", kit],  
          ["living room", liv]]
```

```
# Print out house
```

```
# Print out the type of house
```

- Finish the list of lists so that it also contains the bedroom and bathroom data. Make sure you enter these in order!
- Print out house; does this way of structuring your data make more sense?
- Print out the type of house. Are you still dealing with a list?



```
# house information as list of lists
house = ["hallway", hall],
        ["kitchen", kit],
        ["living room", liv],
        ["bedroom",bed],
        ["bathroom",bath]]
```

```
# Print out house
print(house)
```

```
# Print out the type of house
print(type(house))
```



Exercise - 4

```
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0,  
"bedroom", 10.75, "bathroom", 9.50]
```

Print out the second element from the areas list, so 11.25.

Subset and print out the last element of areas, being 9.50.
Using a negative index makes sense here!

Select the number representing the area of the living room
and print it out.



Exercise - 4

```
# Create the areas list
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom",
10.75, "bathroom", 9.50]

# Print out second element from areas
print(areas[1])

# Print out last element from areas

print(areas[-1])

# Print out the area of the living room
print(areas[5])
```

Exercise -5



Create the areas list

```
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0,  
"bedroom", 10.75, "bathroom", 9.50]
```

Using a combination of list subsetting and variable assignment, create a new variable, `eat_sleep_area`, that contains the sum of the area of the kitchen and the area of the bedroom.

Print the new variable `eat_sleep_area`.



Exercise

```
# Create the areas list
areas = ["hallway", 11.25, "kitchen", 18.0, "living room",
20.0, "bedroom", 10.75, "bathroom", 9.50]
```

```
# Sum of kitchen and bedroom area: eat_sleep_area
eat_sleep_area=areas[3]+areas[7]
```

```
# Print the variable eat_sleep_area
print(eat_sleep_area)
```



Exercise -6

```
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0,  
"bedroom", 10.75, "bathroom", 9.50]
```

Use slicing to create a list, downstairs, that contains the first 6 elements of areas.

Do a similar thing to create a new variable, upstairs, that contains the last 4 elements of areas.

Print both downstairs and upstairs using print().



```
# Create the areas list
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0,
"bedroom", 10.75, "bathroom", 9.50]
```

```
# Use slicing to create downstairs
downstairs=areas[0:6]
```

```
# Use slicing to create upstairs
upstairs=areas[6:10]
```



List slicing (selecting parts of lists)

 [Python youtube playlist](#)



Using the third "step" argument

```
lst = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']  
lst[::2]  
# Output: ['a', 'c', 'e', 'g']  
lst[::3]  
# Output: ['a', 'd', 'g']
```



Selecting a sublist from a list

```
lst = ['a', 'b', 'c', 'd', 'e']
```

```
lst[2:4]
```

Output: ['c', 'd']

```
lst[2:]
```

Output: ['c', 'd', 'e']

```
lst[:4]
```

Output: ['a', 'b', 'c', 'd']



[Python youtube playlist](#)



Reversing a list with slicing

```
a = [1, 2, 3, 4, 5]
# steps through the list backwards (step=-1)
```

```
b = a[::-1]
# built-in list method to reverse 'a'
```

```
a.reverse()
```

```
if a == b:
    print(True)
    print(b)
```

```
# Output:
# True
# [5, 4, 3, 2, 1]
```

[Python youtube playlist](#)

```
1 a=[10,20,30,'kk','kk']
2 print(type(a))
3 print(a[0:3])
4 print(a[::-1])
5 print(a)
6 a.reverse()
7 print(a)
8 print(len(a))
9 print(a[7])
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

Variable explorer Help Plots Files

Console 1/A

```
>
In [8]: runfile('C:/Users/matam/.spyder-py3/untitled0.py', wdir='C:/Users/matam/.spyder-py3')
<class 'list'>
[10, 20, 30]
['kk', 'kk', 30, 20, 10]
[10, 20, 30, 'kk', 'kk']
['kk', 'kk', 30, 20, 10]
5
Traceback (most recent call last):
  File "C:/Users/matam/.spyder-py3/untitled0.py", line 9, in <module>
    print(a[7])
IndexError: list index out of range
```

- index based operation will not effect the behaviour of list true/false
- behaviour of list will effected if we use class list methods ? True/false
- can we declare empty list ?
- does list support arithmetic operations ?
- how to check size of list ?
- does list/set/dict/tuple support only homogeneous ?
- When you get index error ?
- When you will get value error ?
- When you will get type error?
- Can we have duplicate elements in list & set?



Set Data Type

Created on Thu Apr 8 07:44:42 2021

@author: Divya

"""

```
a={10,20,30,40,50,10,20,30,40,50,60,70,80,90}
```

```
print(type(a))
```

```
In [105]: runfile('C:/Users/Divya/untitled80.py', wdir='C:/Users/Divya')
```

```
<class 'set'>
```

```
In [106]:
```

Data Types



Sets are unordered collections of unique objects, there are two types of set

1. Sets - They are mutable and new elements can be added once sets are defined

```
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}  
print(basket) # duplicates will be removed  
> {'orange', 'banana', 'pear', 'apple'}
```

```
a = set('abracadabra')  
print(a) # unique letters in a  
> {'a', 'r', 'b', 'c', 'd'}
```

```
a.add('z')  
print(a)  
> {'a', 'c', 'r', 'b', 'z', 'd'}
```



[Python youtube playlist](#)



- Set doesn't support index
- Set is mutable
- As they are unordered there is no meaning of indexing
- We can add only single element using add() method
- We can add multiple elements using update() method
- Update() method can take tuples, lists, strings(all cases duplicates are avoided)

TypeError in set(s)

```
b={10,20,30,10,220,30,10,20,30}
print(b)
print(len(b))
print(b[0])
```

Usage

Here you can get help of any object by pressing **Ctrl+H** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

Variable explorer Help Plots Files

Console 1/A

```
<class 'list'>
[10, 20, 30]
['kk', 'kk', 30, 20, 10]
[10, 20, 30, 'kk', 'kk']
['kk', 'kk', 30, 20, 10]
5
['kk', 'kk', 30, 20, 10, 100, 300]

In [14]: runfile('C:/Users/matam/.spyder-py3/untitled1.
Users/matam/.spyder-py3')
{10, 220, 20, 30}
4
Traceback (most recent call last):

  File "C:\Users\matam\.spyder-py3\untitled1.py", line
    print(b[0])
TypeError: 'set' object is not subscriptable
```

TypeError in set(s)

```
ice.py × Parent_Child_Class_Inheritance.py × untitled0.py × untitled1.py ×
b={10,20,30,10,220,30,10,20,30}
print(b)
print(len(b))

a={10,100,101,200,201,201}

b.add(111)
print(b)
b.add(a)
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically by placing a left parenthesis next to an object. You can change this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

Variable explorer Help Plots

Console 1/A ×

```
In [15]: runfile('C:/Users/matam/.spyder-py3/untitled1.py', wdir='C:/Users/matam/.spyder-py3')
{10, 220, 20, 30}
4
{10, 111, 20, 220, 30}

In [16]: runfile('C:/Users/matam/.spyder-py3/untitled1.py', wdir='C:/Users/matam/.spyder-py3')
{10, 220, 20, 30}
4
{10, 111, 20, 220, 30}
Traceback (most recent call last):

  File "C:/Users/matam/.spyder-py3/untitled1.py", line 10, in <module>
    b.add(a)
TypeError: unhashable type: 'set'
```

 [Python youtube playlist](#)

Add list to set data structure

```
b={10,20,30,10,220,30,10,20,30}
print(b)
print(len(b))

a=[10,100,101,200,201,201]

b.add(111)
print(b)
b.update(a)
print(b)
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

Variable explorer Help Plots Files

Console 1/A

```
File "C:\Users\matam\.spyder-py3\untitled1.py", line 9,
    b.add(a)
```

TypeError: unhashable type: 'set'

```
In [17]: runfile('C:/Users/matam/.spyder-py3/untitled1.py'
Users/matam/.spyder-py3')
{10, 220, 20, 30}
4
{10, 111, 20, 220, 30}
{100, 101, 200, 201, 10, 111, 20, 220, 30}
```

```
In [18]: runfile('C:/Users/matam/.spyder-py3/untitled1.py'
Users/matam/.spyder-py3')
{10, 220, 20, 30}
4
{10, 111, 20, 220, 30}
{100, 101, 200, 201, 10, 111, 20, 220, 30}
```



[Python youtube playlist](#)

ValueError in set(s)

```
1 b={10,20,30,10,220,30,10,20,30}
2 print(b)
3 print(len(b))
4
5 a=[10,100,101,200,201,201]
6
7 b.add(111)
8 print(b)
9 b.update(a)
10 print(b)
11
12 a.remove(220)
13 print(a)
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

Variable explorer Help Plots Files

Console 1/A X

```
{10, 220, 20, 30}
4
{10, 111, 20, 220, 30}
{100, 101, 200, 201, 10, 111, 20, 220, 30}
[10, 100, 101, 200, 201, 201]

In [21]: runfile('C:/Users/matam/.spyder-py3/untitled1.
Users/matam/.spyder-py3')
{10, 220, 20, 30}
4
{10, 111, 20, 220, 30}
{100, 101, 200, 201, 10, 111, 20, 220, 30}
Traceback (most recent call last):

  File "C:/Users/matam/.spyder-py3/untitled1.py", line
    a.remove(220)
ValueError: list.remove(x): x not in list
```

KeyError in set(s)

```
b={10,20,30,10,220,30,10,20,30}  
print(b)  
print(len(b))
```

```
a=[10,100,101,200,201,201]
```

```
b.add(111)  
print(b)  
b.update(a)  
print(b)
```

```
b.remove(220)  
print(b)  
b.remove(320)  
print(b)
```

Usage

Here you can get help of any object by pressing **Ctrl+H** in front of it, either on the Editor or the Console.

Help can also be shown automatically after a left parenthesis next to an object. You can adjust this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

Variable explorer Help Plots Files

Console 1/A x

```
b.remove(220)
```

KeyError: 220

```
In [27]: runfile('C:/Users/matam/.spyder-py3/untitled1.py',  
              Users/matam/.spyder-py3')
```

```
{10, 220, 20, 30}
```

```
4
```

```
{10, 111, 20, 220, 30}
```

```
{100, 101, 200, 201, 10, 111, 20, 220, 30}
```

```
{100, 101, 200, 201, 10, 111, 20, 30}
```

Traceback (most recent call last):

```
File "C:/Users/matam/.spyder-py3/untitled1.py",
```

```
b.remove(320)
```

KeyError: 320



[Python youtube playlist](#)



How to remove elements from set ?

discard()

remove() - raises error if really the element doesn't exists (Key error)

Pop() – this method also used to remove element from set (Uncertainty due to unordered data)

```
ss={10,20,34,405,60,60,53,43,30,20,10,"nagesh","nagesh"}
print(ss)
ss.discard(600)
print(ss)
ss.pop()
ss.pop()

print(ss)

print("=====")
ll=[10,20,34,405,60,60,53,43,30,20,10,"nagesh","nagesh"]
print(ll)
ll.remove(60)
print(ll)
ll.pop()
ll.pop()
print(ll)
```

1	list	11	[10, 20, 34, 405, 60, 60, 53, 43, 30, 20, ...]
liv	float	1	20.0
ll	list	10	[10, 20, 34, 405, 60, 53, 43, 30, 20, 10]

```
In [47]: runfile('C:/Users/Divya/untitled28.py', wdir='C:/Users/Divya')
{34, 10, 43, 20, 405, 53, 'nagesh', 60, 30}
{34, 10, 43, 20, 405, 53, 'nagesh', 60, 30}
{10, 43, 20, 405, 53, 'nagesh', 60, 30}
=====
[10, 20, 34, 405, 60, 60, 53, 43, 30, 20, 10, 'nagesh', 'nagesh']
[10, 20, 34, 405, 60, 53, 43, 30, 20, 10, 'nagesh', 'nagesh']
[10, 20, 34, 405, 60, 53, 43, 30, 20, 10, 'nagesh']

In [48]: runfile('C:/Users/Divya/untitled28.py', wdir='C:/Users/Divya')
{34, 10, 43, 20, 405, 53, 'nagesh', 60, 30}
{34, 10, 43, 20, 405, 53, 'nagesh', 60, 30}
{43, 20, 405, 53, 'nagesh', 60, 30}
=====
[10, 20, 34, 405, 60, 60, 53, 43, 30, 20, 10, 'nagesh', 'nagesh']
[10, 20, 34, 405, 60, 53, 43, 30, 20, 10, 'nagesh', 'nagesh']
[10, 20, 34, 405, 60, 53, 43, 30, 20, 10]
```

[Python youtube playlist](#)



with other sets

```
# Intersection
{1, 2, 3, 4, 5}.intersection({3, 4, 5, 6}) # {3, 4, 5}
{1, 2, 3, 4, 5} & {3, 4, 5, 6}             # {3, 4, 5}

# Union
{1, 2, 3, 4, 5}.union({3, 4, 5, 6}) # {1, 2, 3, 4, 5, 6}
{1, 2, 3, 4, 5} | {3, 4, 5, 6}      # {1, 2, 3, 4, 5, 6}

# Difference
{1, 2, 3, 4}.difference({2, 3, 5}) # {1, 4}
{1, 2, 3, 4} - {2, 3, 5}           # {1, 4}

# Symmetric difference with
{1, 2, 3, 4}.symmetric_difference({2, 3, 5}) # {1, 4, 5}
{1, 2, 3, 4} ^ {2, 3, 5}                    # {1, 4, 5}

# Superset check
{1, 2}.issuperset({1, 2, 3}) # False
{1, 2} >= {1, 2, 3}          # False

# Subset check
{1, 2}.issubset({1, 2, 3}) # True
{1, 2} <= {1, 2, 3}        # True

# Disjoint check
{1, 2}.isdisjoint({3, 4}) # True
{1, 2}.isdisjoint({1, 4}) # False
```



Set Operations using Methods and Builtins

We define two sets a and b

```
>>> a = {1, 2, 2, 3, 4}
```

```
>>> b = {3, 3, 4, 4, 5}
```

NOTE: {1} creates a set of one element, but {} creates an empty **dict**. The correct way to create an empty set is **set()**.

Method	Operator
<u>a.intersection(b)</u>	<u>a & b</u>
<u>a.union(b)</u>	<u>a b</u>
<u>a.difference(b)</u>	<u>a - b</u>
<u>a.symmetric_difference(b)</u>	<u>a ^ b</u>
<u>a.issubset(b)</u>	<u>a <= b</u>
<u>a.issuperset(b)</u>	<u>a >= b</u>

 [Python youtube playlist](#)



Intersection

`a.intersection(b)` returns a new set with elements present in both a and b

```
>>> a.intersection(b)
{3, 4}
```

Union

`a.union(b)` returns a new set with elements present in either a and b

```
>>> a.union(b)
{1, 2, 3, 4, 5}
```

Difference

`a.difference(b)` returns a new set with elements present in a but not in b

```
>>> a.difference(b)
{1, 2}
>>> b.difference(a)
{5}
```



Symmetric Difference

`a.symmetric_difference(b)` returns a new set with elements present in either `a` or `b` but not in both

```
>>> a.symmetric_difference(b)
```

```
{1, 2, 5}
```

```
>>> b.symmetric_difference(a)
```

```
{1, 2, 5}
```

NOTE: `a.symmetric_difference(b) == b.symmetric_difference(a)`

Subset and superset

`c.issubset(a)` tests whether each element of `c` is in `a`.

`a.issuperset(c)` tests whether each element of `c` is in `a`.

```
>>> c = {1, 2}
```

```
>>> c.issubset(a)
```

```
True
```

```
>>> a.issuperset(c)
```

```
True
```



2. Frozen Sets - They are immutable and new elements cannot added after its defined.

```
b = frozenset('asdfagsa')
```

```
print(b)
```

```
> frozenset({'f', 'g', 'd', 'a', 's'})
```

```
cities = frozenset(["Frankfurt", "Basel", "Freiburg"])
```

```
print(cities)
```

```
> frozenset({'Frankfurt', 'Basel', 'Freiburg'})
```

frozenset() – Immutable set

```
s={}
s={10,20,30,10,20,30}
print(s)
s.add(100)
print(s)

si=frozenset(s)
print(si)
si.add(100)
```

libraryName
ipython dtype asyncio

Help Variable explorer Plots File

Console 1/A x

```
{10, 20, 30}
{100, 10, 20, 30}

In [228]: runfile('C:/Users/Divya/untitled94.py', wdir='C:
{10, 20, 30}
{100, 10, 20, 30}
frozenset({100, 10, 20, 30})

In [229]: runfile('C:/Users/Divya/untitled94.py', wdir='C:
{10, 20, 30}
{100, 10, 20, 30}
frozenset({100, 10, 20, 30})
Traceback (most recent call last):

  File "C:/Users/Divya/untitled94.py", line 16, in <module>
    si.add(100)

AttributeError: 'frozenset' object has no attribute 'add'
```



Set Membership Test

```
# initialize my_set  
my_set = set("apple")
```

```
# check if 'a' is present  
# Output: True  
print('a' in my_set)
```

```
# check if 'p' is present  
# Output: False  
print('p' not in my_set)
```




Creating empty set

```
emptySet = set()
```

```
dataScientist = set(['Python', 'R', 'SQL', 'Git', 'Tableau', 'SAS'])  
dataEngineer = set(['Python', 'Java', 'Scala', 'Git', 'SQL', 'Hadoop'])
```

To remove all variables from set

```
Setdata.clear()
```



Remove Duplicates from a List

Approach 1: Use a set to remove duplicates from a list.

```
print(list(set([1, 2, 3, 1, 7])))
```

Approach 2: Use a list comprehension to remove duplicates from a list

```
def remove_duplicates(original):  
    unique = []  
    [unique.append(n) for n in original if n not in unique]  
    return(unique)
```

```
print(remove_duplicates([1, 2, 3, 1, 7]))
```

Inbuilt functions in python

1. `print()`
2. `type()`
3. `list()`
4. `set()`
5. `frozenset()`
6. `len()`