Data structures in Python

Dr. S.Chellammal

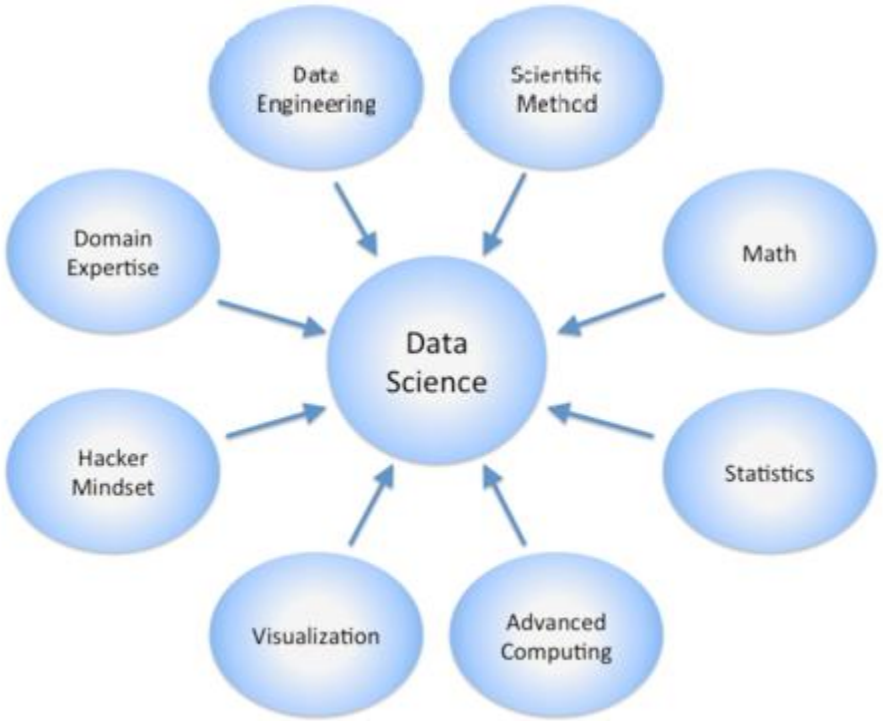
29.05.2018

Data science

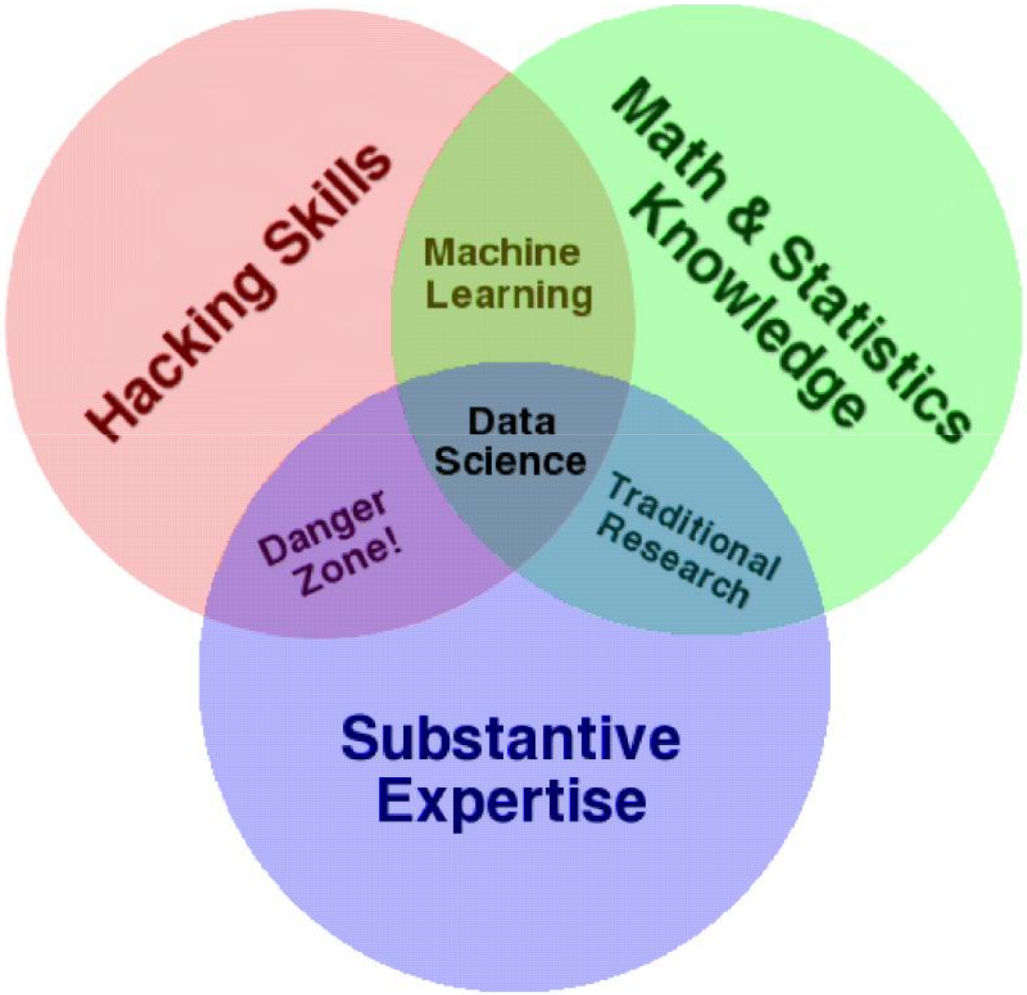
- Data science is a broader term - compiling and analyzing data in order to present decision to management.
- A data scientist's ultimate goal - to discover new knowledge. In business, these insights may mean a huge edge for the company.
- Or it may mean a breakthrough in current methods, like a brand new analysis technique.
- Or it may mean a different paradigm altogether;
- the data scientist discovers how to apply existing techniques in a novel way

Deeper goal

- Demands to draw meaningful insights in a multifaceted manner. Implies many areas



Data science



Data science

- Internet search: Search engines make use of data science algorithms to deliver best results for search queries in a fraction of seconds.

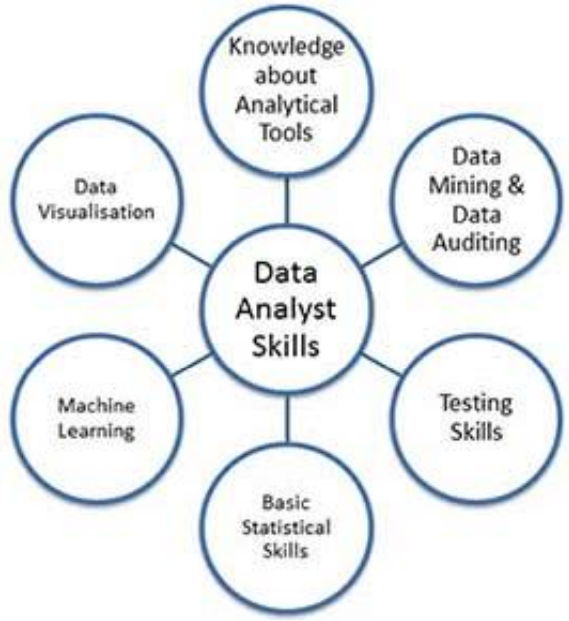
- Digital Advertisements: The entire digital marketing spectrum uses the data science algorithms - from display banners to digital billboards. This is the mean reason for digital ads getting higher CTR than traditional advertisements.

- Recommender systems: The recommender systems not only make it easy to find relevant products from billions of products available but also adds a lot to user-experience

Data analytics

- *Data Analytics:* Data Analytics the science of examining raw data with the purpose of drawing conclusions about that information.
- Data Analytics involves applying an algorithmic or mechanical process to derive insights. For example, running through a number of data sets to look for meaningful correlations between each other.

Data analyst skills



Programming skills
Statistical skills
Machine learning skills
Data wrangling skills – conversion of data in one form to another which allows more convenient consumption of data
Communication and visualization skills

Data analytics

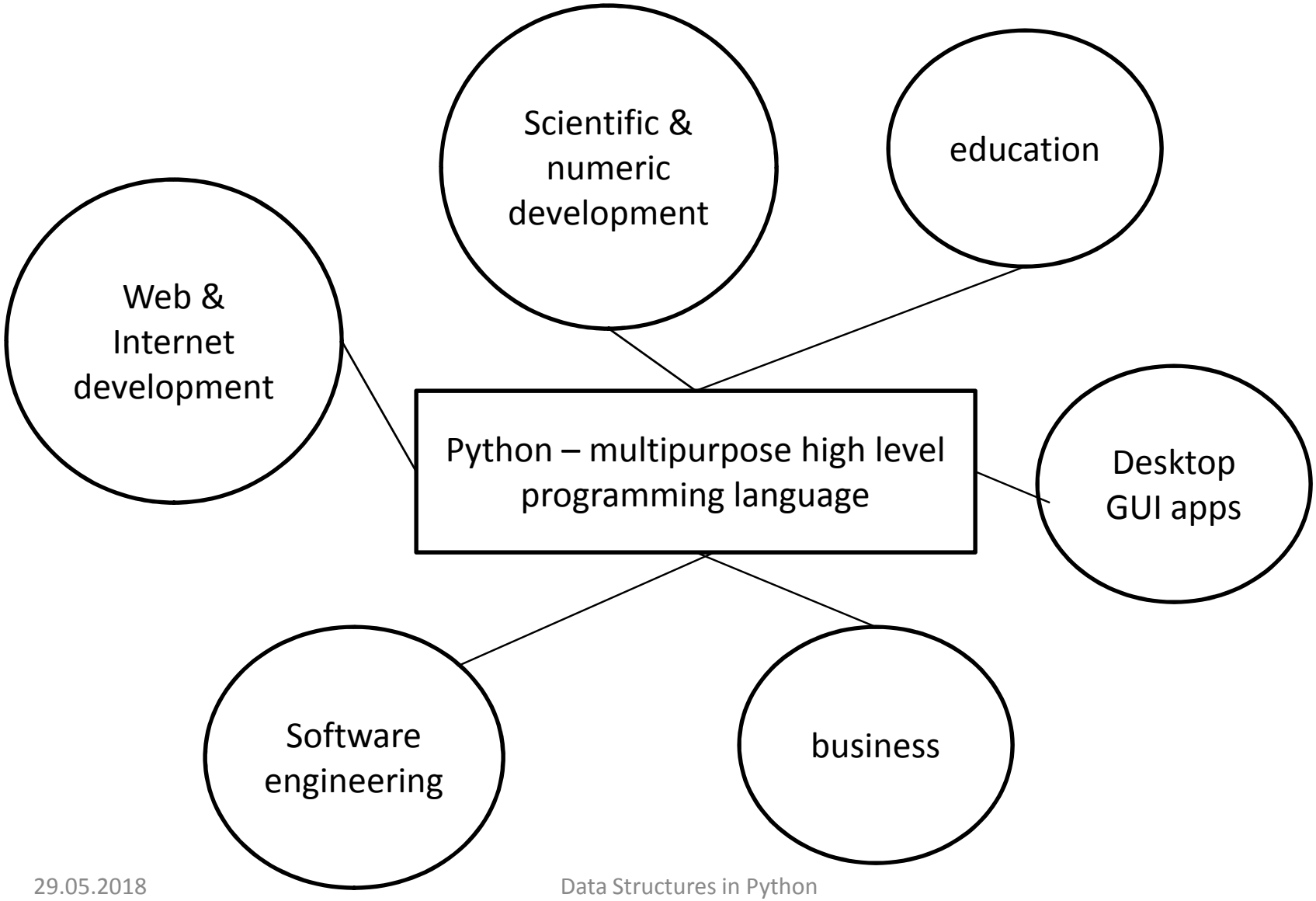
- More specific and confined term
- Refers to processes, techniques, methods to analyze data(regardless of the size of the data)

- Focus is to draw meaningful insight from the data
- Automating the meaningful insights into certain datasets & usages of queries and data aggregation procedures

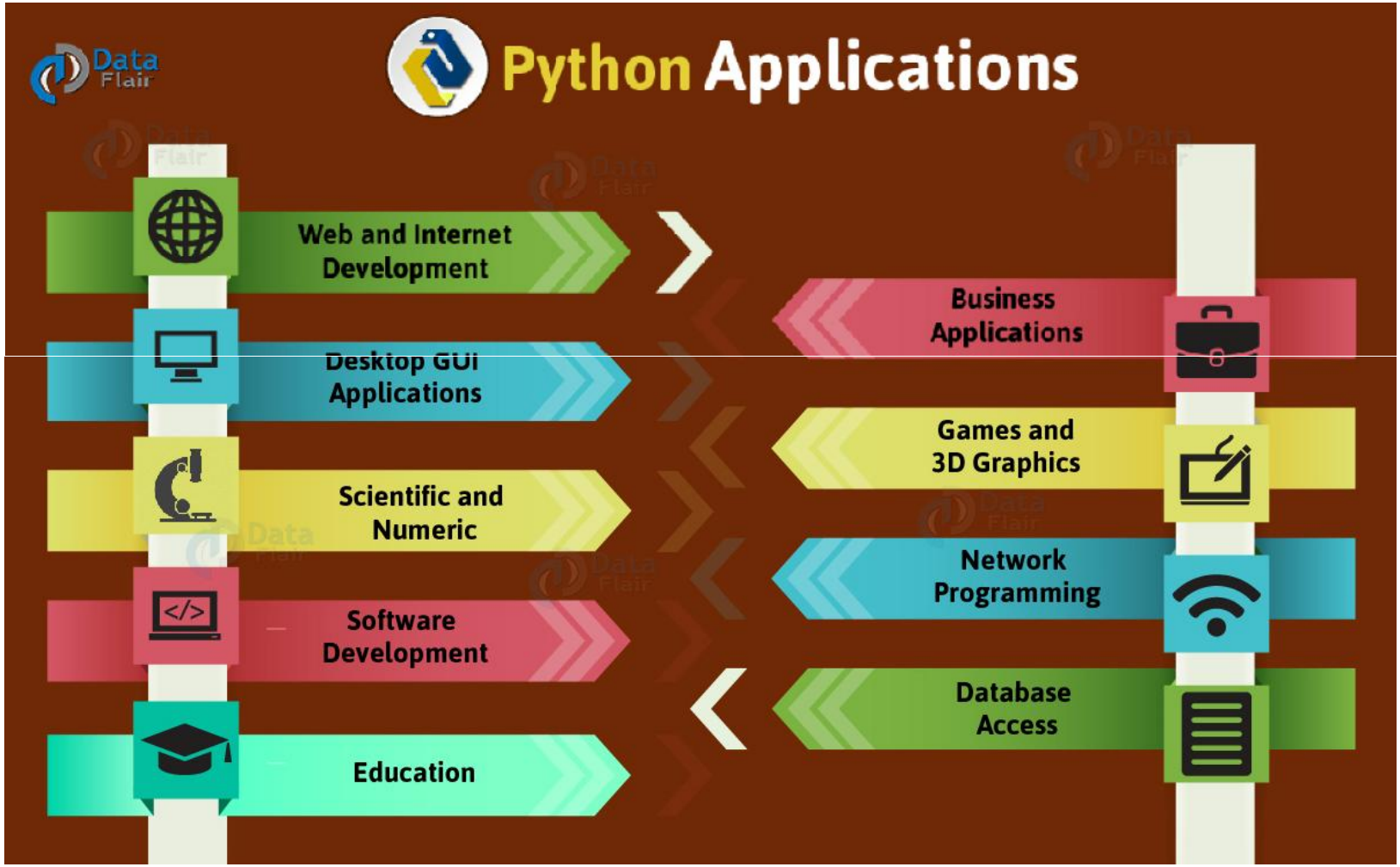
Example areas

- Healthcare
- Travel
- Gaming
- Energy management

Python


Scientific & numeric development
education
Web & Internet development
Python – multipurpose high level programming language
Desktop GUI apps
Software engineering
business

Python applications



Python packages for different domains

- Web & Internet app development
  - Django, Pyramid – web framework – quick development with less code
  - Flask, bottle – micro frameworks
  - Support Internet protocols – HTML,XML, Email, IMAP, FTP,
  - Requests – client package
  - beautifulsoup-HTML parser
  - Twisted python – asynchronous networking

Scientific Python

- Scipy – for maths, science & engineering apps
- Numpy – Numerical python
- Pandas – for data analysis
- Ipython – interactive shell & editing & recording work sessions, visualization & parallel computing

GUI development

- Tk
- Kivy – developing multitouch apps
- Wxwidgets

Software development

- Scons – build packags
- Buildbot, Apache Gump – for compiling & testing
- Roundup – bug tracking
- Trac – project management

Business app

- ERP & Ecommerce apps
- Odoo – all in one package supports a wide range of developing business apps

- Tryton – three tier app platform for developing business apps

Why python

- East to use
- Simple & more readability
- Less code
- Interpreted
- Comprehensive & large set of libraries
- Modular
- Portable
- Object oriented
- Easy integration
- Improved programmer’s productivity
- Free & Open source

limitations

- Weak support for mobile computing
- Weak support for db connectivity(not like JDBC, ODBC)
- Dynamically typed – so, more testing time is required & errors at final run
- slow

Who developed python?

- Guido Van Rossum
- Python Software Foundation – an organization holds the copyright of Python versions
- Python versioning major.minor.micro

Python installation

- Python version 3.4.2 installer
- Installs
- numpy-1.9.2-win32-superpack-python3.4

- For pandas
- Pip install pandas (requires .net 4 framework & microsoft visual studio 10/12)

Python Libraries for Data analytics

Many popular Python toolboxes/libraries:

- NumPy
- SciPy
- Pandas
- SciKit-Learn

Visualization libra

- matplotlib
- Seaborn

All these libraries are installed on the SCC

and many more …

Python Libraries for Data Science

NumPy:

- introduces objects for multidimensional arrays and matrices, as well as functions that allow to easily perform advanced mathematical and statistical operations on those objects

- provides vectorization of mathematical operations on arrays and matrices which significantly improves the performance

- many other python libraries are built on NumPy

Python Libraries for Data Science

SciPy:

- collection of algorithms for linear algebra, differential equations, numerical integration, optimization, statistics and more
- part of SciPy Stack
- built on NumPy

Python Libraries for Data Science

Pandas:

- adds data structures and tools designed to work with table-like data (similar to Series and Data Frames in R)
- provides tools for data manipulation: reshaping, merging, sorting, slicing, aggregation etc.
- allows handling missing data

Python Libraries for Data Science

SciKit-Learn:

- provides machine learning algorithms: classification, regression, clustering, model validation etc.

- built on NumPy, SciPy and matplotlib

Python Libraries for Data Science

matplotlib:

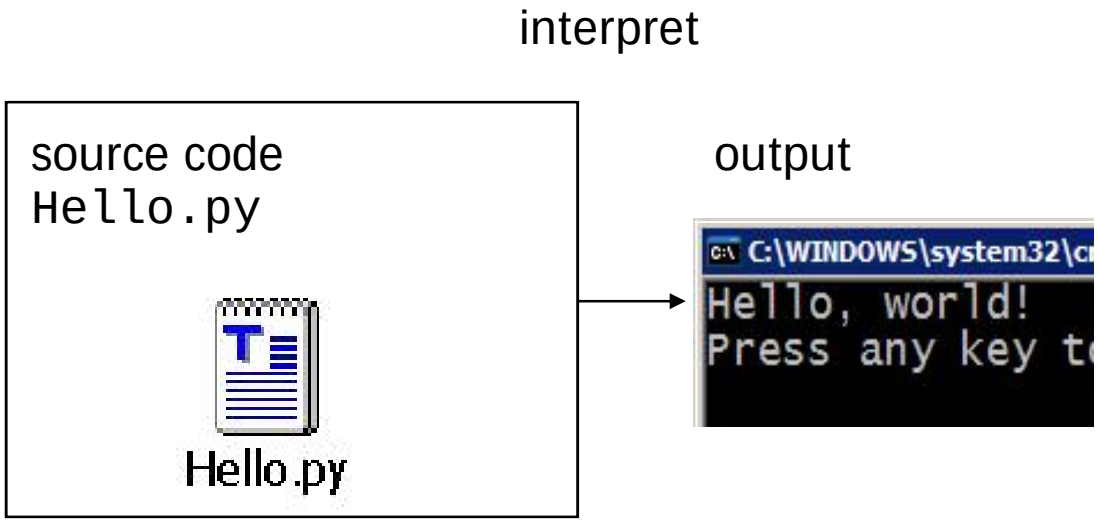
- python 2D plotting library which produces publication quality figures in a variety of hardcopy formats
- a set of functionalities similar to those of MATLAB
- line plots, scatter plots, barcharts, histograms, pie charts etc.
- relatively low-level; some effort needed to create advanced visualization

Python Libraries for Data Science

Seaborn:

- based on matplotlib
- provides high level interface for drawing attractive graphics
- Similar to the popular ggplot2 library in R

Python - interpreted language



Major Python versions

- “Python” or “CPython” is written in C/C++
  - Version 2.7 came out in mid-2010
  - Version 3.1.2 came out in early 2010

- “Jython” is written in Java for the JVM
- “IronPython” is written in C# for the .Net environment

Development environment

1. PyDev with Eclipse
2. Komodo
3. Emacs
4. Vim
5. TextMate
6. Gedit
7. Idle
8. PIDA (Linux)(VIM Based)
9. NotePad++ (Windows)
10. BlueFish (Linux)

two modes

- Command mode
- Script mode

Command mode

```
print(“God is great”)

print(‘god is great’)

i=10
print(i)
print(“the value of i ”, i)
```

Script mode

- Open IDEL
- File new
- Save with .py
- C:\python d:\python34\test.py

range

- The `range` function specifies a range of integers:
  - `range(`***start***`, `***stop***`)` - the integers between ***start*** (inclusive) and ***stop*** (exclusive)

  – It can also accept a third value specifying the change between values.
    - `range(`***start***`, `***stop***`, `***step***`)` - the integers between ***start*** (inclusive) and ***stop*** (exclusive) by ***step***

  – Example:

```
for x in range(5, 0, -1):
    print x
```

Output:

```
5
4
3
2
1
```

for loop

```
sum = 0
for i in range(1, 11):
    sum = sum + (i * i)
print "sum of first 10 squares is", sum

Output:
sum of first 10 squares is 385
```

while loop

While condition:

    statements

Example

```
i=1
While i in range(1,11)
    print(i)
    i=i+1
```

Selection (if-elif-else)

```
a=6
b=7
If(a<b):
   pri nt(“a is less than b”)
elif(a>b):
   print(“a is greater than b”)
else:
   print(“a is equal to b”)
```

input

```
a=input("enter the value of a")
print("\n")
b=input("enter the value of b")
print("\n")
numa=int(a)
numb=int(b)
```

import

```
>>> import math
math.cos(0)
>>> from math import cos, pi
Cos(0)
>>> from math import *
cos(0)
```

math module

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.cos(0)
1.0
>>> math.cos(math.pi)
-1.0
>>> dir(math)
['__doc__', '__file__', '__name__', '__package__', 'acos', 'acosh',
'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos',
'cosh', 'degrees', 'e', 'exp', 'fabs', 'factorial', 'floor', 'fmod',
'frexp', 'fsum', 'hypot', 'isinf', 'isnan', 'ldexp', 'log', 'log10',
'log1p', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan',
'tanh', 'trunc']
>>> help(math)
>>> help(math.cos)
```

math module

Command name	Description
abs(value)	absolute value
ceil(value)	rounds up
cos(value)	cosine, in radians
floor(value)	rounds down
log(value)	logarithm, base e
log10(value)	logarithm, base 10
max(value1, value2)	larger of two values
min(value1, value2)	smaller of two values
round(value)	nearest whole number
sin(value)	sine, in radians
sqrt(value)	square root

function

- def add(x,y):

  return(x+y)

Function invoking

add(6+8)

Store it in a file and import

myclass1.py

```
class FirstClass:
    def display(self):
        print("god is great")

myemp=FirstClass()
myemp.display()
```

myclass2.py

```
class Employee:
    def __init__(self,id,name,salary):
        self.id=id
        self.name=name
        self.salary=salary
    def display(self):
        print(self.id)
        print(self.name)
        print(self.salary)

emp1=Employee('1001','murugan',10000)
emp1.display()
```

myclass3.py

```
class Employee:
    def __init__(self,id,name,salary,account):
        self.id=id
        self.name=name
        self._salary=salary
        self.__account=account
    def display(self):
        print(self.id)
        print(self.name)
        print(self._salary)
        print(self.__account)

emp1=Employee('1001','murugan',10000, '234678-234')
emp1.display()
```

exception

```
import sys
a=int(input("enter value for a"))

b=int(input("enter value for b"))


try:
    print(a/b)
    print("after exception")
    filename=sys.argv[1]
    f=open(filename,"r")
    data=f.readlines()
    print(data)
    f.close()
except ZeroDivisionError:
    print("division by zero invalid: Re-enter b")

except ValueError:
    print("string cannot be converted into interger")

except IOError:
    print("cannot open the mentioned file")
```

string

- Sequence of characters

When you type a character, it is converted into ASCII or Unicode number – encoding

Similarly ascii to letters is decoding

Data structures - string

```
my_string = 'Hello‘
print(my_string)
my_string = "Hello“
print(my_string)
my_string = '''Hello'''‘
print(my_string)
```

```
>>> print('''this is a string extends for more than one line.
    starting with triple quotes and endding with triple quotes''')
this is a string extends for more than one line.  starting with
    triple quotes and endding with triple quotes
>>>
```

accessing string using index

```
str = 'programiz‘
print('str = ', str)

#first character
print('str[0] = ', str[0])

#last character
print('str[-1] = ', str[-1])
```

Some errors while accessing

```
# index must be in range
>>> my_string[15] ...
IndexError: string index out of range
 # index must be an integer
>>> my_string[1.5] ...
TypeError: string indices must be integers
```

Slicing - : operator

```
#slicing 2nd to 5th character

print('str[1:5] = ', str[1:5])

#slicing 6th to 2nd last character

print('str[5:-2] = ', str[5:-2])
```

Strings are immutable

```
>>> mystr="hello good morning"
>>> mystr[1]='a'
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    mystr[1]='a'
TypeError: 'str' object does not support item assignment
>>>
```

The same is true for del(mystr[1])

```
TypeError: 'str' object doesn't support item deletion
```

String concatenation

```
>>> str="hello world"
>>> str[-1]
'd'
>>> str[5:]
' world‘
>>> str[:5]
'hello'
>>> str[0:5]+" good "+str[5:]
'hello good  world'
>>> print(str)
hello world
>>>
```

String concatenation – parenthesis multiline example

```
>>> mystr=('this is in first line'
... 'in second line'
... 'in thired line'
... 'in fourth line')
>>> print(mystr)
this is in first linein second linein thired linein
   fourth line
>>>
```

String/text processing in a for loop

```
>>> for i in x:
    print(i)


c
h
e
l
s
>>>
```

String membership

```
x=‘chels’
>>> 'e' in x
True
>>> 'k' in x
False
>>> 'k' not in x
True
>>>
```

Common string methods

```
>>> len(x)
5
>>>
>>> x.upper()
'CHELS‘
>>> x.replace('s','la')
'chella'
>>>
>>> x.find('e')
2
 >>> str.rfind('l')
4     #returns the last index of l
>>>
```

find() & index() difference

```
>>> str.find('s')   # if not found -1.  example
-1
>>> >>> x.index('t')
Traceback (most recent call last):
  File "<pyshell#51>", line 1, in <module>
    x.index('t')
ValueError: substring not found
```

count() method

```
>>> str="python is simple.  is it not?"
>>> substr='is'
>>> str.count(substr)
2
>>> >>> len(str)
29
>>> str.count(substr,12,28)
1
>>>
```

split method

```
>>> str="hello,world"
>>> b=str.split(',')
>>> print(b)
['hello', 'world']
>>> >>> print(b[1])
world
>>> print(b[0])
hello
>>> >>> b[0]="hai"
>>> b[1]="good world"
>>> print(b)
['hai', 'good world']
>>>
```

Some more string methods

str.isalnum() - String consists of only alphanumeric characters (no symbols)

str.isalpha()- String consists of only alphabetic characters (no symbols)

str.islower() - String's alphabetic characters are all lower case

str.isnumeric() - String consists of only numeric characters

str.isspace() String consists of only whitespace characters

str.istitle()String is in title case

str.isupper()String's alphabetic characters are all upper case

array

an array is a data structure that stores values of same data type.

array is not a fundamental data type in Python

we have to import ‘array’ module

Creating array

```
>>> import array
>>> myarray1=array.array('i',[10,11,12,13,14])
>>> print(myarray1)
array('i', [10, 11, 12, 13, 14])
>>> for x in myarray1: print(x)

10
11
12
13
14
>>>
```

Accessing individual elements

```
>>> mya=array.array("i",[10,20,30,40,50])
>>> mya[0]=100
>>> print(len(mya))
5
>>> mya[4]=500
>>> print(mya)
array('i', [100, 20, 30, 40, 500])
>>>
```

Insert & append

```
>>> mya.insert(0,50)
>>> print(mya)
array('i', [50, 100, 20, 30, 40, 500])
>>> mya.append(600)
>>> print(mya)
array('i', [50, 100, 20, 30, 40, 500, 600])
>>>
```

count

```
>>> print(mya)
array('i', [50, 100, 20, 30, 40, 500, 600])
>>> mya.append(100)
>>> mya.count(100)
2
>>>
```

extend

To extend elements with another array

```
>>> myb=[10,20,30]
>>> mya.extend(myb)
>>> print(mya)
array('i', [50, 100, 20, 30, 40, 500, 600, 100, 10,
   20, 30])
>>>
```

To add elements from list

```
>>> a=[1000,2000,3000]
>>> mya.fromlist(a)
>>> print(mya)
array('i', [50, 100, 20, 30, 40, 500, 600, 100, 10,
   20, 30, 1000, 2000, 3000])
>>>
```

To add from string

```
>>> mystr=array.array('u',['c','h'])
>>> print(mystr)
array('u', 'ch')
>>> mystr.fromunicode("els")
>>> print(mystr)
array('u', 'chels')
>>>
```

pop & remove

```
>>> m=array.array('i',[1,2,3,4])
>>> m=array.array('i',[1,2,3,4])
>>> m.pop()
4
>>> m.remove(3)
>>> print(m)
array('i', [1, 2])
>>>
```

reverse

```
>>> m=array.array('i',[1,2,3,4,5,6,7,8,9,10])
>>> m.reverse()
>>> for x in m: print(x)

10
9
8
7
6
5
4
3
2
1
>>> print(m)
array('i', [10, 9, 8, 7, 6, 5, 4, 3, 2, 1])
>>>
```

tostring, tolist

```
>> b=myc.tostring()
>>> print(b)
b'c\x00h\x00e\x00l\x00s\x00'
>>> myc.tolist()
['c', 'h', 'e', 'l', 's']
>>>
```

Array slicing

```
>>> import array
>>> a=array.array('i',[1,2,3,4]
... )
>>> a[:3]
array('i', [1, 2, 3])
>>> a[-1]
4
>>> a[1:]
array('i', [2, 3, 4])
>>>
```

list

A single list can contain strings, integers, as well as objects. It can contain heterogeneous data.

Lists are mutable, i.e., they can be altered once declared.

```
>>> mylist=[1,2,3,'chels',4.6]
>>> print(mylist)
[1, 2, 3, 'chels', 4.6]
>>>
```

Same methods as in array

- Appending
- Inserting
- Slicing
- Removing
- Updating
- Finding element
- Clear(), pop(),remove(),reverse(),sort()

Some real examples with list

```
def linearSearch(myList, myItem):
   position=0
   found='false'
   while(position<len(myList) and found=='false'):
      if(myList[position]==myItem):
         found='true'
      position=position+1
   return found

x=[1,2,3,4,5,6,7,8,9]
y=31
retval=linearSearch(x,y)
print("the value is: ", retval)
```

binarysearch

```
def binarySearch(myList,myItem):
    found='false'
    bottom=0
    top=len(myList)-1
    if(bottom<top):middle=(bottom+top)//2
    print("the middle value is: ",middle)

    if(myList[middle]==myItem): found='true'
    if(myList[middle]<myItem):
        print("at this place - upper half")
        bottom=middle+1
        while(bottom<top and found=='false'):
            print("at this place - upper loop")
            if(myList[bottom]==myItem): found='true'
            bottom=bottom+1
    if(myList[middle]>myItem):
        print("at this place - lower half")
        top=middle-1
        bottom=0
        while(bottom<top and found=='false'):
            print("at this place - lower loop")
            if(myList[bottom]==myItem): found='true'
            bottom=bottom+1
    return found
```

Binary search contd.

```
a=[12,13,14,15,16,17,22,28,30,36,80]
b=90
retval=binarySearch(a,b)
print(retval)
```

bubblesort

```
def bubbleSort(myList):
    end=len(myList)
    for i in range(end):
        for j in range(i+1, end):
            if(myList[j]<myList[i]):
                myList[j],myList[i]=myList[i],myList[j]
    return myList

a=[4,78,6,23,90,12,34,7,46,673,12,23,34,45,67,123,12,4,0]
retval=bubbleSort(a)
print(retval)
```

Find unique elements from list uniques.py

```
ef unique(list1):

    # intilize a null list
    unique_list = []

    # traverse for all elements
    for x in list1:
        # check if exists in unique_list or not
        if x not in unique_list:
            unique_list.append(x)
    # print list
    for x in unique_list:
        print(x)
    a=set(unique_list)
    print(a)


l=[10,20,30,40,10,20]
unique(l)
```

Remove,replace some elements

Remove some

a[0:3]=[]

Clear all

a[:]=[]

```
>>> a=[4,5,6,7,8,9]
>>> a[0:3]=[1,2,3]
>>> print(a)
[1, 2, 3, 7, 8, 9]
>>>
```

Array versus list

Array	**List**
Homogeneous	Heterogeneous
Empty array cannot be created	Empty list can be created
Arithmetic operations can be done	No arithmetic operations can be done
Literature says memory storage is less comparatively	Python list is a heterogeneous list, the list in memory stores references to objects rather than the number themselves.

List or lists (list2d.py)

```
mylist=[[10,20,30],[40,50,60]]
mylist[0][0]
for i in range(0,2,1):
    for j in range(0,3,1):
        print(mylist[i][j])
```

tuple

- A tuple is created by placing all the items (elements) inside a parentheses (), separated by comma.

- A tuple can have any number of items and they may be of different types (integer, float, list, string etc.).
- Tuple is immutable

len funcion

```
>>> t=('a','p','p','l','e')
>>> len(t)
5
>>>
```

Creating tuple

Empty tuple

```
>>> t=()
>>> print(t)
()
```

Creating tuple of one element

```
>>>  t=(1)
>>> print(t)
1
>>> t=(1,)
>>> print(t)
(1,)
>>>
```

Tuple is heterogeneous

```
>>> myt=('chels',0.456,100,'c')
>>> print(type(myt))
<class 'tuple'>
>>>
>>> t=('a','p','p','l','e')
>>> len(t)
5
```

Accessing elements in tuple

```
>>> my_tuple = (4, 2, 3, [6, 5])
>>> my_tuple[0]
4
>>> my_tuple[:2]
(4, 2)
>>> my_tuple[0:2]
(4, 2)
>>> my_tuple[-1]
[6, 5]
>>> my_tuple[3][0]
6
>>> my_tuple[3][1]
5
>>> my_tuple[3]
[6, 5]
>>>
```

Tuple - immutable

```
>>> my_tuple[1]=9
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    my_tuple[1]=9
TypeError: 'tuple' object does not support item
   assignment
>>>
```

But delete is possible

```
>>> del(my_tuple)
>>> my_tuple
Traceback (most recent call last):
  File "<pyshell#35>", line 1, in <module>
    my_tuple
NameError: name 'my_tuple' is not defined
>>>
```

Tuple - concatenation

- >>> my_tuple=my_tuple+(6,7,8)
- >>> print(my_tuple)
- (4, 2, 3, [6, 5], 6, 7, 8)
- >>> my_tuple
- (4, 2, 3, [6, 5], 6, 7, 8)
- >>>

Tuple methods

```
>>> myt=('a','p','p','l','e')
>>> myt.count('p')
2
>>> myt.index('p')
1
>>> myt.index('t')
Traceback (most recent call last):
  File "<pyshell#39>", line 1, in <module>
    myt.index('t')
ValueError: tuple.index(x): x not in tuple
>>>
```

Membership test in tuple

```
>>> 'a' in myt
True
>>>
```

Iterating tuple

```
>>> for item in
   ('chels','murugan','sri','ammu'):print(item)


chels
murugan
sri
ammu
>>>
```

all,any

```
>>> mya=(1,2,3)
>>> print(all(mya))
True
>>> mya=(1,2,3)+(0,)
>>> print(mya)
(1, 2, 3, 0)
>>> print(all(mya))
False
```

- Similarly any

sorted

```
>>> sorted(mya)
[0, 1, 2, 3]
>>>
>>> sorted(mya)
[0, 1, 2, 3]
>>> b=sorted(mya)
>>> b[0]=10
>>> b[0]
10
```

enumerate

```
>>> myt=(60,70,80,90)
>>> e=enumerate(myt)
>>> c=list(e)

>>> for index,item in c: print(index,item)
...
0 60
1 70
2 80
3 90
>>>
```

Packing and unpacking tuple

Packing is a simple syntax which lets you create tuples "on the fly" without using the normal notation:

a = 1, 2, 3

Unpacking

x, y, z = a

set

Set in Python is unordered collection of unique elements.

the order of elements in a **set** is undefined.

Sets are mutable

One can add and delete elements of a **set**

one can iterate the elements of the **set**,

one can perform standard operations on **sets** (union, intersection, difference).

Set - creation

```
a = {}
# check data type of a
# Output: <class 'dict'>
print(type(a))

my_set = {1, 2, 3}
print(my_set)
# set of mixed datatypes
my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)
>>> print(a)
{}
>>> print(type(a))
<class 'dict'>
>>> a=set()
>>> print(type(a))
<class 'set'>
>>>
```

Adding elements to set

```
>>> a.add(1,2,3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: add() takes exactly one argument (3 given)
>>> a.add(1)
>>> a.add(2)
>>> a.add(3)
>>> print(a)
{1, 2, 3}
>>> a.add(1)
>>> print(a)
{1, 2, 3}
>>>
```

Set operations

- >>> a={1,2,3,4}
- >>> b={3,4,5,6}
- >>> print(a|b) #union
- {1, 2, 3, 4, 5, 6}
- >>> print(a&b) #intersection
- {3, 4}
- >>> print(a-b) #difference
- {1, 2}
- >>> print(b-a) #difference
- {5, 6}
- >>>

Set – symmetric difference

```
>>> a={1,2,3,4}
>>> b={3,4,5,6}
>>> print(a^b)
{1, 2, 5, 6}
>>>
```

Clear

```
>>> a.clear()
>>> a
set()
>>>
```

isdisjoint

If there is no common elements between two sets

```
>>> print(a)
{1, 2}

>>> print(b)
{3,4,5,6}
>>> print(a.isdisjoint(b))
True
>>>
```

subset

```
>>> a={1,2,3,4}
>>> b={1,2}
>>> b.issubset(a)
True
>>> a.issubset(b)
False
>>>
```

remove, discard

```
>>> a.discard(1)
>>> print(a|b)
{2, 3, 4, 5, 6}
>>> a.discard(10)
>>> a.remove(10)
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    a.remove(10)
KeyError: 10
>>> a.remove(3)
>>> print(a)
{2, 4}
>>>
```

A real application

Cartesian product of sets

A = {1,2} and B = {4,5,6}

AxB = {(1,4),(1,5),(1,6),(2,4),(2,5),(2,6)}

A = {a,b} and B = {2,9}

AxB = {(a,2),(a,9),(b,2),(b,9)}

Cartesian product helps in say purchase of a cars

Problem

Consider that you sell cars and a customer comes to you with the following request.

“I need a number of cars. The models I want are Toyota Corolla, Honda CRV, and Chevy Cruze.

I need my cars to be white, grey, and red and

I want them to have 1.4, 1.6, and 1.8 engine capacities.

I want a combination of cars that satisfy the specifications I’ve stated.”

Find the cars with above spec

- M = {Toyota Corolla, Honda CRV, Chevy Cruze}
- C = {White, Grey, Red}
- E = {1.4L, 1.6L, 1.8L}

Car combinations

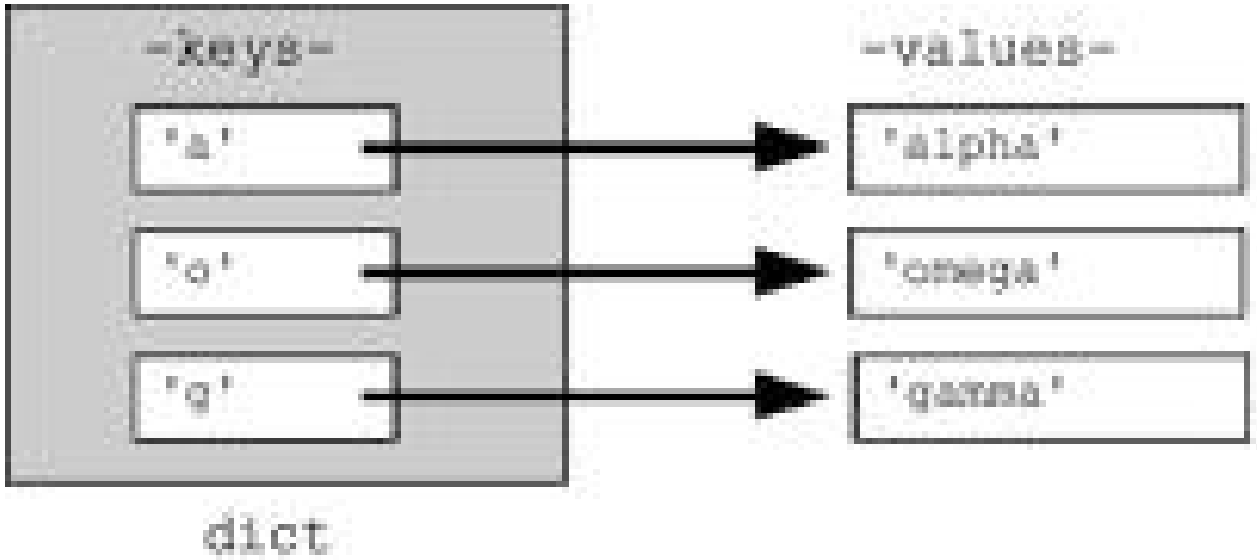
```
models=['TC','HCRV','Chevy Cruze']
colors=['white','grey','red']
capacities=['1.4','1.6','1.8']
products=[]
for model in models:
    for color in colors:
        for capacity in capacities:
            products.append([model,color,capacity])



for item in products: print(item)
```

dictionary

- A **dictionary** is an associative array.
- Any key of the **dictionary** is associated (or mapped) to a value. Heterogeneous data type
- dictionaries are unordered key-value-pairs.



Creating dictionary

```
>>> d=dict()
>>> d={}                    #or d={‘a’:’india,’b’:’srilanka}
>>> d['a']='india'
>>> d['b']='srilanka'
>>> print(d)
{'b': 'srilanka', 'a': 'india'}
>>>
>>> len(d)
2
>>>
```

Printing keys & values

- >>> print(d.keys())
- dict_keys(['b', 'a'])
- >>> print(d.values())
- dict_values(['srilanka', 'india'])
- >>>

Printing the value of a particular key

```
>>> print(d)
{'b': 'malaysia', 'u': 'china', 'a': 'india'}
>>> d.get('a')
'india'
>>>
```

Printing the items – dict view

```
>>> print(d.items())
dict_items([('b', 'malaysia'), ('u', 'china'), ('a',
   'india')])
>>>
```

Clear the dictionary

```
>>> d.clear()
>>> print(d)
{}
>>>
```

Iteration over dictionary

```
>>> e=enumerate(d)
>>> print(e)
<enumerate object at 0x023F1710>
>>> for index, value in e:
    print(index,value,d[value])


0 b srilanka
1 a india
>>>
```

Check whether a particular key exists

```
>>> print('a' in d)
True
>>> print('u' in d)
False
>>>
```

Adding items to dict & update

```
>>> d['u']='china'
>>> print(d)
{'b': 'srilanka', 'u': 'china', 'a': 'india'}
>>>
>>> d['b']='malaysia'
>>> print(d)
{'b': 'malaysia', 'u': 'china', 'a': 'india'}
>>>
```

Populating keys

```
>>> mykeys=('1','2','3','4','5')
>>> d.fromkeys(mykeys)
{'4': None, '5': None, '1': None, '3': None, '2': None}
>>> d['1']='a'
>>> d['2']='e'
>>> d['3']='i'
>>> d['4']='o'
>>> d['5']='u'
>>> print(d)
{'4': 'o', '5': 'u', '1': 'a', '3': 'i', '2': 'e'}
>>>
```

Merging two dictionaries(using update mehod)

```
>>> d1={'1':'a','2':'e'}
>>> d2={'3':'i','4':'o','5':'u'}
>>> d1.update(d2)
>>> print(d1)
{'1': 'a', '2': 'e', '3': 'i', '4': 'o', '5': 'u'}
>>>
```

Keys are unique, it picks up latest value assigned to it

```
>>> d3={'1':'a','2':'an','3':'the'}
>>> print(d3)
{'1': 'a', '2': 'an', '3': 'the'}

>>> d3['3']='some'
>>> d3['4']='the'
>>> print(d3)
{'1': 'a', '2': 'an', '4': 'the', '3': 'some'}
>>>
```

Dictionary to list (dictolist.py)

```
dict = {}
dict['Capital']="London"
dict['Food']="Fish&Chips"
dict['2012']="Olympics"

#lists
temp = []
dictList = []


for key, value in dict.items():
   temp = [key,value]
   dictList.append(temp)

print (dictList)
for i in dictList:
   print (i)
```

stack

Stack = Last in First Out data structure
Can be implemented using List. (methods- append and pop)

```
stack = ["Amar", "Akbar", "Anthony"]
stack.append("Ram")
stack.append("Iqbal")
print(stack)
print(stack.pop())
print(stack)
print(stack.pop())
print(stack)
```

queue

First In First Out

```
>>> from collections import deque
>>> myq=deque(['a','b','c'])
>>> print(myq)
deque(['a', 'b', 'c'])
>>> myq.append('d')
>>> myq.append('e')
>>> myq.popleft()
'a'
>>>
```

Advantages of numpy

Numpy provides a high-performance multidimensional array object, and tools for working with these arrays.

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers.

A list is the Python equivalent of an array, but contain elements of different types.

A common beginner question is what is the real difference here. The answer is performance.

Numpy data structures perform better in:

Size - Numpy data structures take up less space

Performance - they have a need for speed and are faster than lists

Functionality - SciPy and NumPy have optimized functions such as linear algebra operations built in.

How?

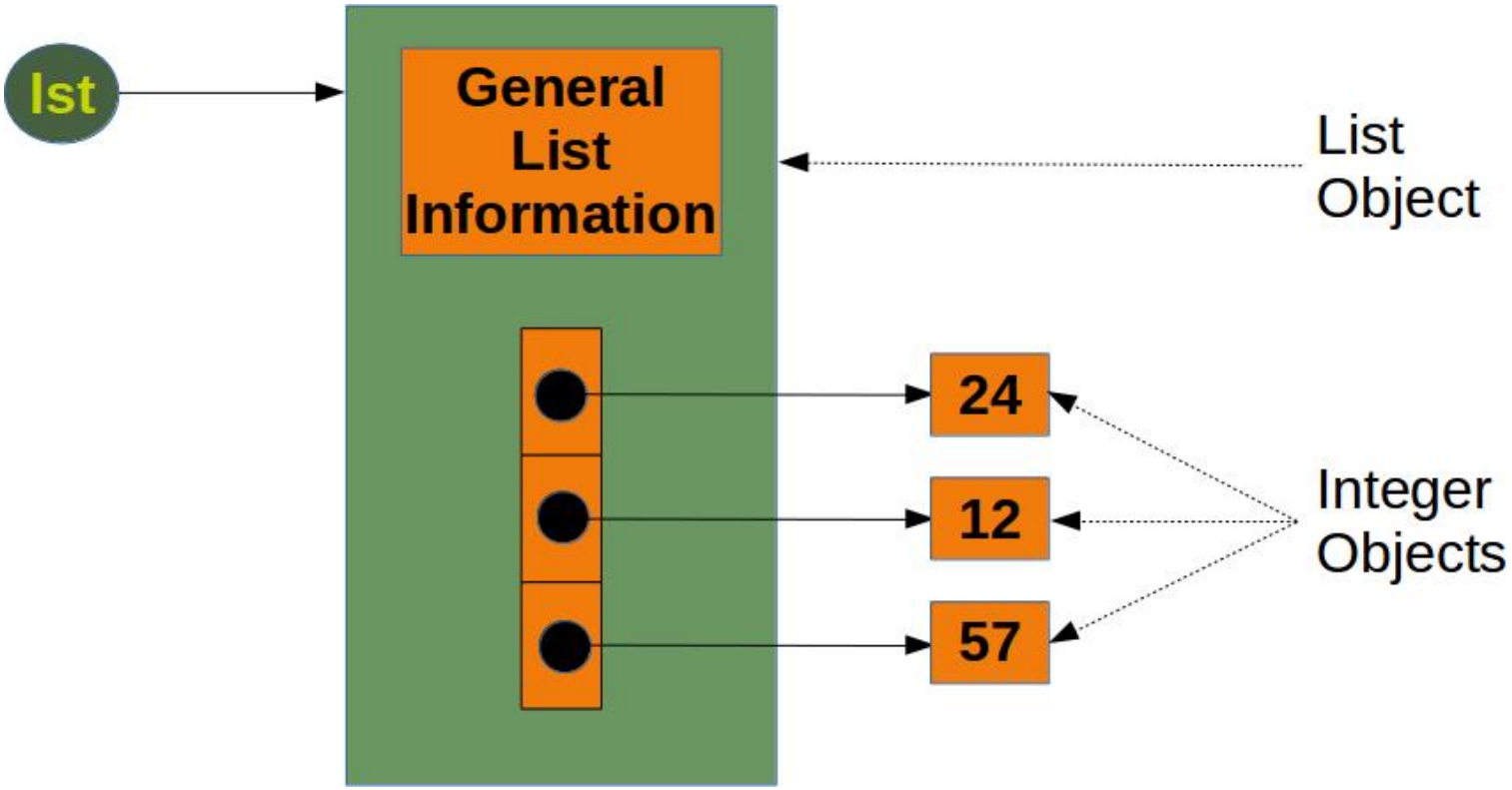
Numpy offers multidimensional array – homogeneous data can be stored.

Indexing is with numbers.

But list requires object references to processing the items of list

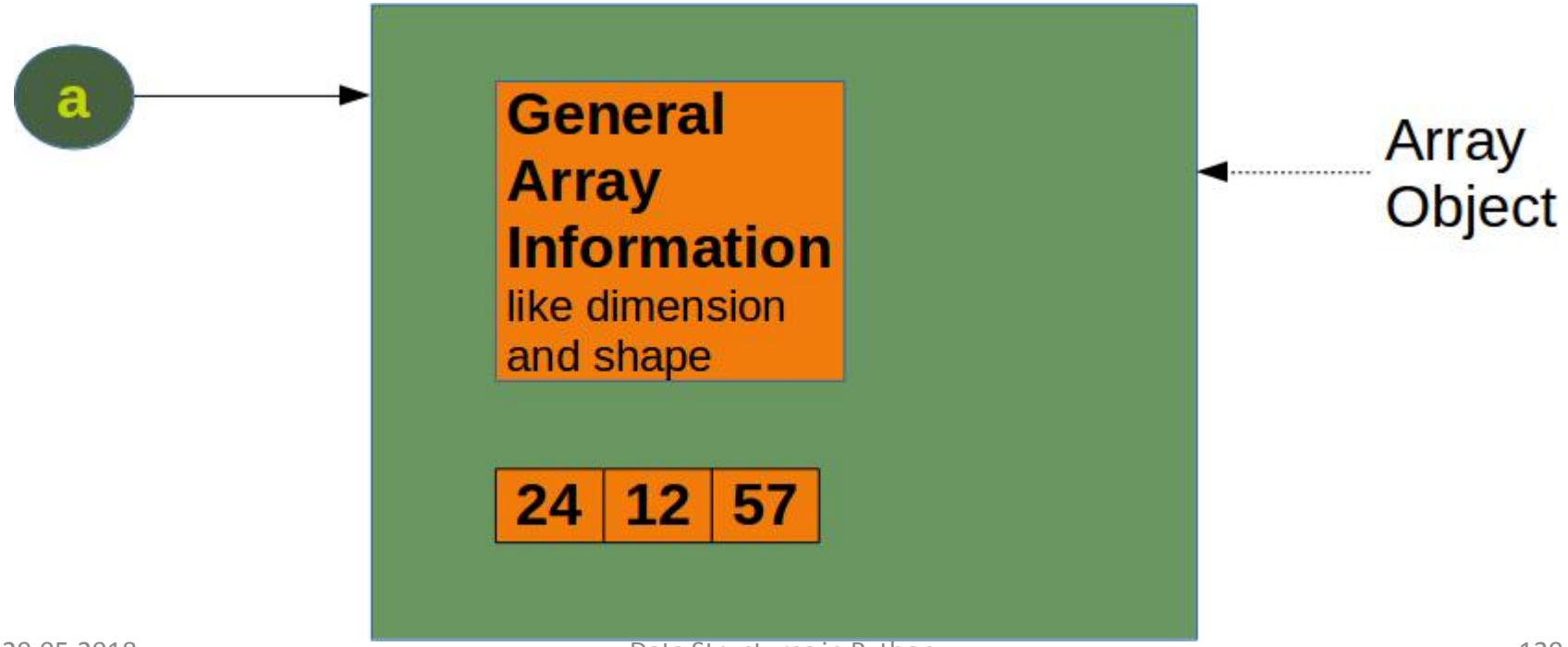
Memory consumption by list

Bytes in memory = 64 + 8 * len(lst) + *+ len(lst)* * 28



Memory consumption by numpy array

- NumPy takes up less space. This means that an arbitrary integer array of length "n" in numpy needs. Memory = 96 + n * 8 Bytes

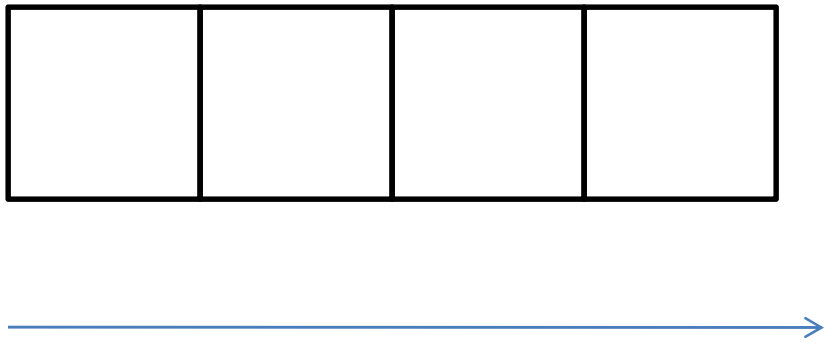


numpy

Number of dimensions of array – rank

Size of array along each dimension is shape

Axis 0 – one dimensional array



Axis 0

Accessing & modifying elements from 1D array

```
>>> array1D=np.array([1,2,3])
>>> for i in array1D: print(i)
...
1
2
3
>>> array1D[0]=4
>>> array1D[1]=5
>>> array1D[2]=6
>>> print(array1D)
[4 5 6]
>>>
```

Sum along axis 0

```
>>> array1D.sum(axis=0)
15
>>> array1D.ndim
1
>>> array1D.shape
(3,)

>>> array1D.dtype
dtype('int32')
```

Creation of 2D array

Axis 1

2 (0,0)	4 (0,1)	6 (0,2)
3 (1,0)	6 (1,1)	9 (1,2)
4 (2,0)	8 (2,1)	12 (2,2)
5 (3,0)	10 (3,1)	15 (3,2)

Axis 0

Sum along axis 0 = [14,28,42] (columnwise sum)

Sum along axis 1=[12,18,24,30] (rowwise sum)

Creating 2Darray using numpy

```
>>> array2D=np.array([[2,4,6],[3,6,9],[4,8,12],[5,10,15]])
>>> print(array2D)
[[ 2  4  6]
 [ 3  6  9]
 [ 4  8 12]
 [ 5 10 15]]
>>> array2D.ndim
2
>>> array2D.dtype
dtype('int32')
>>> array2D.shape
(4, 3)
>>>
```

Sum along axis0 & axis1

```
>>> array2D.sum(axis=0)
array([14, 28, 42])  # column-wise sum

>>> array2D.sum(axis=1)
array([12, 18, 24, 30])   # row-wise sum
>>>
```

Accessing elements of 2Darray

```
>>> print(array2D[2,0])
4
>>> array2D[2,0]=8
>>> print(array2D[2,0])
8
>>>
```

Printing all elements in array2D

Numpy2d.py

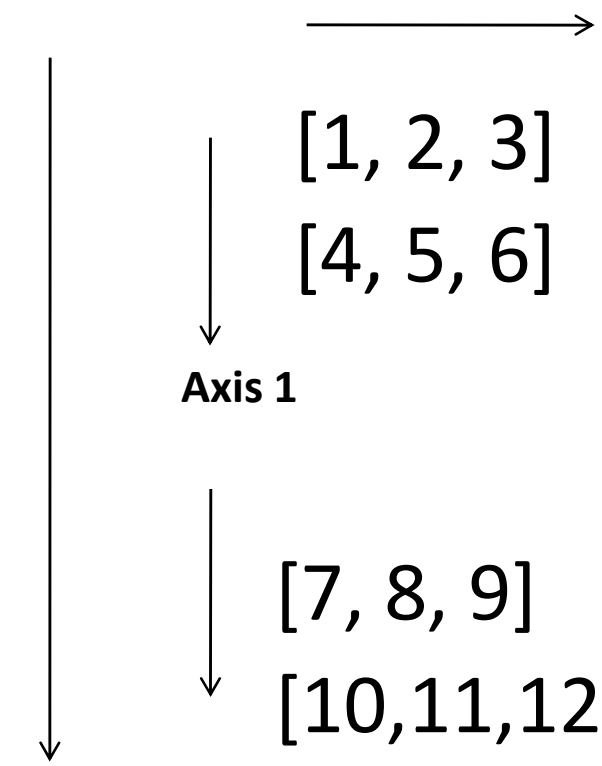
```
import numpy as np
array2D=np.array([[2,4,6],[3,6,9],[4,8,12],[5,10,15]])
for i in range(array2D.shape[0]):
    for j in range(array2D.shape[1]):
        print(array2D[i,j])
```

Creating 3d

```
>>> import numpy as np
>>> array3D=np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])
>>> print(array3D)
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]]
>>> array3D.ndim
3
>>> array3D.shape
(2, 2, 3)
>>>
```

3d axes



[1, 2, 3]

[4, 5, 6]

axis 2 = rowwise

axis 1 = columnwise

[7, 8, 9]

[10,11,12]

axis 0 – 1+7,2+8,3+9

4+10,5+11,6+12

Sum axis 0, axis 1, axis 2

```
>>> array3D.sum(axis=2)
array([[ 6, 15],
       [24, 33]])
>>> array3D.sum(axis=1)
array([[ 5,  7,  9],
       [17, 19, 21]])
>>> array3D.sum(axis=0)
array([[ 8, 10, 12],
       [14, 16, 18]])
>>>
```

Accessing elements from array3D

```
>>> print(array3D[0,0,0])
1
>>> print(array3D[1,1,2])
12
>>>
```

Printing all elements in array3D numpy3d.py

```
import numpy as np
array3D=np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])
for i in range(array3D.shape[0]):
    for j in range(array3D.shape[1]):
        for k in range(array3D.shape[2]):
            print(array3D[i,j,k])
```

slicing

```
>>> myarray=array3D[0:1,0:2,0:3]
>>> print(myarray)
[[[1 2 3]
  [4 5 6]]]
>>>
```

Zeros, ones

```
>>> myzero=np.zeros(2,2)
>>> myzeros=np.zeros((2,2))
>>> print(myzeros)
[[ 0.  0.]
 [ 0.  0.]]
>>> myones=np.ones((2,2))
>>> print(myones)
[[ 1.  1.]
 [ 1.  1.]]
```

diagonal

```
>>> mydias=np.diag(np.array([1,2]))
>>> print(mydias)
[[1 0]
 [0 2]]
>>>
>>> print(np.eye(3))
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
>>>
```