Quick Guide to Python for C programmer

For full Python library documentation, you should always reference https://www.python.org/doc/

Do note: Version 3.x is NOT backward compatible with Version 2.x

Last update : July 30th, 2018

CONTENTS

SCOPE

1. This document assumes you have already had a sound foundation in C, a bit of knowledge in OOPS is helpful.
2. This document guides a quick alignment guide to most commonly used syntax between Python and C, not meant to be a full blown tutorial.
3. Far more importantly, you should learn how to be resourceful to look up the latest update.
4. Will focus more on the data structure design and their usage.
5. Do note that version 3 is not compatible with version 2. Therefore, you will need to always refer back to the official Python Documentation - https://docs.python.org .

How to use this document:
- Go thru each section and try out the samples
- Exercises:
- Be inquisitive and question how and why
- There are plethoric amount of online reference, such as:
- https://www.tutorialspoint.com/python/ for more beginners
- http://www.geeksforgeeks.org/python/ for more savvy programmers

SET UP :

INSTALL PYTHON 3.X ONTO YOUR WINDOWS SYSTEM:

1) Download Python 3.6.1 : https://www.python.org/downloads/windows/ - Windows x86 web-based installer
2) After installation, make sure you know where your have installed it at. In most cases, it is installed at:

   C:\Program Files (x86)\Python36-32

3) edit setvar.bat:

```
set PYTHONPATH="C:\Program Files (x86)\Python36-32"
set PYTHONPATH=%PYTHONPATH%;%PYTHONPATH%\libs;%PYTHONPATH%\Lib
rem DLL  %PYTHONPATH%\DLLs

set path=%path%;%PYTHONPATH%\bin
```

4) Since we are going to work with numpy module, you should install now.

```
cd %PYTHONPATH%\Scripts
pip3  install  numpy
```

5) Configuration files that you should be aware of. You do not need to customize it. See www.python.org for details if you want to customize it.

py.ini, pyvenv.cfg, PYTHONHOME, PYTHONPATH

6) Change python code to executable ?

Cx_Freeze is a distutils extension (see Extending Distutils) which wraps Python scripts into executable.

7) Byte-compile Python libraries

- Compile a single file : python -m (Lib/py_compile.py) - to generate a byte-code file from a source file
- Compile multiple files : python -m compileall

(Lib/compileall.py)

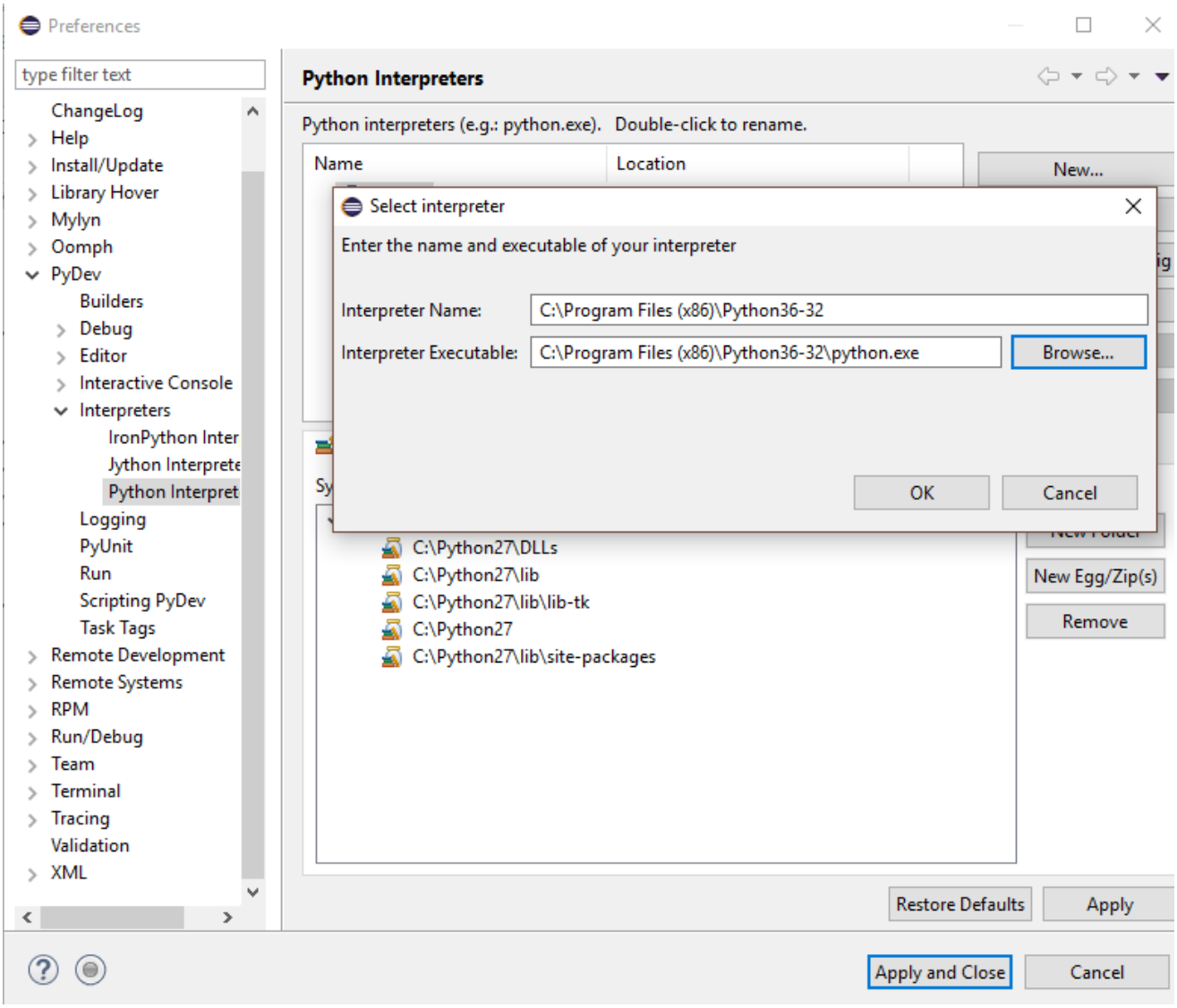
I'll add more into individual modules installation later. For more detailed information, you may refer to :

https://docs.python.org/2/installing/index.html

IDE for Python

- Visual Studio 2015 and up
- Eclipse plugin:
    - Install Eclipse Neon
    - go to http://marketplace.eclipse.org/marketplace-client-intro?mpc_install=114 for installing the plug-in.
- GEdit – a platform just for python
- On Linux/Window: you may use Thonny .

If you use Eclips, you need to set it up the interpreter. Don't forget to click on Apply and close.



INSTALL PYTHON 3.X ONTO YOUR PI

Working with Command line

Install 3.4 : sudo apt-get install python3

- (so that you don't have to rebuild)

Install 3.6 : https://gist.github.com/dschep/24aa61672a2092246eaca2824400d37f

- Let's not to do this, as you will have to rebuild the whole package. It will take a while.

RUN IT

```
$  python3
```

Check out the version. It should be 3.4+

Working with Gui IDE

```
$ sudo apt-get install python3-thonny
```

To run it:

```
$ thonny
```

To put it on your desktop:

Option 1: Use the Xsession window

- Simply click on the raspberry icon, and find the add shortcut from selection.

Option 2: Put a soft link in your Desktop folder:

```
cd ./Desktop
#  ln -s /path/to/target_file /path/to/symlink
e.g.
which thonny  # to find where it is installed
ln -s /usr/bin/thonny  ./PythonThonny
```

Option 3:

Edit a file in your Desktop folder with the following:

```
[Desktop Entry]
Name="the name that displays under the icon on the desktop"
Exec="a single command with or without arguments to execute on double click"
Icon="/path/to/some/128pxX128px.png"
Comment="If I'm here i show in properties tab or something"
```

Sample:

```
[Desktop Entry]
Name="Python 3.6 - Thonny"
Type=Application
Exec="/usr/bin/thonny"
Icon="/home/pi/icons/picons.png"
Comment="Thonny with Pi 3.6.x"
```

Find icon that you like online and download it.:

- E.g. http://icons.iconarchive.com/icons/cornmanthe3rd/plex/32/Other-python-icon.png
- Wget <the url>

Very Useful Links

Build-in Library functions:
https://docs.python.org/3/library/functions.html
About Constants :
https://docs.python.org/2/library/constants.html

Official Python Documentation:
https://docs.python.org/2/index.html
Basic describes syntax and language Reference:
https://docs.python.org/2/reference/index.html

MOST BASIC INTRINSIC TYPES IN PYTHON

You should always refer to the official document to see the full list, and the latest update. https://docs.python.org.

LIST OF MOST BASIC DIFFERENCES BETWEEN C/C++ AND PYTHON

Here lists major differences in most commonly used language structure between Python and C/C++.

Few Major difference:

- Python is an interpreter, not a complied language to executable like C/C++.
- Everything in Python is about “object” (Yes, OOPS)
- No more { } but indentation
- Scope rule only applies within functions, anything outside of that is global
- Unbound variables : variable do not support explicit declaration statement

For those who are familiar with C++:

- Similar support for class and iterators. Will not cover this in this document. Here is the link to more dive-in reading.

C/C++	Python
In Windows, you need the installation folder in your system environment variable $PATH	In linux/unix, you need this at top of a file: #!/usr/bin/python
*.c or *.cpp	*.py
Explicit Pointers reference	Almost everything in Python is in “object” reference. No explicit pointer reference.
int x = 10;	x = 10
single quote is different from double quote char *x = ‘hello’; NOOOO! char *x = “hello”; int a=1, b=2;	single quote is treated the same as double quote e.g. x = ‘hello’ is the same as x= “hello” a,b,x = 1,2,"hello"
void main()…	Do not need it. Expressions have no preceding space are considered to be in “main” per se. So, do this instead in order to simulate scope rule within the main() Creating a function called main: def main:

	the main body At the bottom of the program: main() IMPORTANT: No indentation.
{… }	Indentation only All statements within a block must be indented with the same amount of space.

Global vs local with simple { } scope rule
 	
int var=10;
void func() {
var=20;
printf("%d,", var);
}
void main(){
func();
printf("%d", var);
}
>> 20, 20
	All variables/objects inside a function must be local. If you want to refer to the global, you must use:
	var=10
	def func():
	var = 20
	print (var)
	
	func()
	print (var)
	>> 20, 10
	//----------
	def func():
	global var
	var = 20
	print (var)
	
	you will get
	>> 20, 20

while (I < 10) { …. }
for (i=0; i<10; i++)
printf("%d.", i);
	while (I < 10) :
	for i in range(0,10):
	print (i, end=".")
	note: if you want no line break:
	print(i, end="")
	(range does not support float)

switch ( ) { case : … }	No such implementation. However, you can simulate the effect using dictionary (a Python data structure type)
#include <stdio.h> # not default	Import sys # default
// comment /* block of comment */	# comment ''' block of comment. Either single or double quote will do… '''

primitive data type: int, float, long, char	There is really no such thing as primitive data types. The most fundamental types are in objects context. So, they are all like pointer- reference. So, you can delete them. These are some Python's standard data types: — Numeric types (they are treated like reference too): int, float, long (e.g. 5199L), complex (e.g. 4.5j) — Sequences: String, byte array (3), list, tuple — Sets and mappings: List, Dictionary
Casting float f = (float)10/(float)3	 f = float(10/3)
Array int lookup[6] = {0,1,1,2,3,5}; lookup[5]	from array import * lookup = array('i', [0,1,1,2,3,5]) lookup[5] or lookup[-1]
#include	From module import * e.g. if you want a c/c++ type of array implementation, you need to include the following : from array import *
You cannot delete a variable. However, you can get the same effect using scope rule, i.e. {… } { int varX = 1; }	varX = 1 del varX
x = x + 10 + 20 + 30;	x = x + 10 \ + 20 \ + 30;
Functions… e.g. int factorial(num) {… } void displayResult(char *arr)	No function return type. Just use "def" e.g. def factorial(num): you code here must be indented return result

{ … }	def displayResult(arr) the function code return
char days[][10] = {“Monday”, “Tuesday”, “Wednesday”, “Thursday”, “Friday”};	days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
char sentence[] = “It is a sentence”;	sentence = "It is a sentence."
int counter = 10; printf(“%d\n”, x);	Counter = 10 Print counter + '\n'
printf("\nPress Enter to exit."); while (getchar() != ‘\n’) ;	input("\nPress Enter to exit.") This input can be changed into the data type needed as well. See the sample after this table.
char x[] = “foo”; printf(“%s\n”, x);	x = 'foo'; sys.stdout.write(x + '\n')
printf(“%s age is %d”, name, num);	print “%d age is %d”, name, num,

char x[ ] = “abcde”;
while (*x!=0)
{ printf(“%c,”, *x);
x++;
}
	mylist = “abcde”;
	
	for p in mylist:
	print p,
	or
	print ",".join([str(p) for p in mylist])
	
	do note:
	mylist2 = "abcdefg", output will be *a, b, c, d, e, f, g*
	but
	mylist2 = [*"abcdefg"*], output will be abcdefg

	numbers = [1,10,20,30,40,50]
	sum = 0
	for number in numbers:
	sum = sum + numbers
	print sum
	
	for i in range(1,10, 2):

	print (i) in range(1, 11) creates an object known as an iterable that a

else if (i==1)
…
else
	elif i==2:
	…
	else:

int a = 1, b = 2; char *c = “Larry”;	a, b, c = 1, 2, "Larry"
Void func int func(bool x=true)	def func( ): def func(x=true):
int i=5; sizeof(int) >> you will get 2, or 4 or 8 depending on the platform	import sys i=5 sys.getsizeof(i) you will get 28 or more depending on the platform.
-INCLUDE_PATH	You can do this in your code: sys.path.insert(0, '/home/pi/wk/pinclude')
printf(“float: %.2f”, f);	print ("float: %.2f" % f)
&&	and
||	or
! (	not

Operators in Python do not exist in C or C++

Power: ** e.g. 5**2 =25 B = B**2 B **=2
Floor Division: // e.g. 10 // 3 = 3 X = X//3 X //=3
Membership:

is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

```
 e.g.                a = 10
b = 20
mylist = [1, 2, 3, 4, 5 ]

if   a in list :
     print a +  ‘ is in mylist’

if   a not in list:
     print a, “ is not in mylist”
```

! First step : check your python version

```
python -V
or
python3 -V
```

Simple Standard Input

Get user input as a string

```
#!/usr/bin/python3     # user version 3
// sample code – test.py
import sys

def userInput():
      name = input("Your Name: ")
      print (name)
      return

if sys.version_info[0] < 3:
   raise Exception("Python 3 or a more recent version is required.")

userInput()
```

To run it : python3 test.py

==========

For version 2.x

```
   name = raw_input("Your Name? ")

print name
```

to run it : python test.py

==========

Console: Green is your input. Note that input data has to conform to the python data type syntax.

```
Your Name? "Elizabeth"
Enter [ list of your favourite fruit ] : [orange, apple, kiwi, watermelon]
Elizabeth 's favourite fruit are:  [orange, apple, kiwi, watermelon]
```

Read a list of numbers from user and split all numbers into a "list"

numbers = input("enter numbers: ")	Being read in as a string	enter numbers: 4 94 0
print(numbers)		4 94 0
print (numbers.split())	Split by space(s)	['4', '94', '0']
Print (len(numbers))		3

numbers = input("enter numbers: ")		enter numbers: 4, 94, 0
nl = list (map(int,numbers.split(','))) print (nl)	Split by space(,) and create a list "nl"	[4, 94, 0]

numbers = input("enter numbers: ")		enter numbers: 4, 94, 0
print(numbers)		4 94 0
print (numbers.split())		['4', '94', '0']

```
print (numbers.split())

Enter a list of ints:   12, 24,    1, 6,   90
 [12, 24, 1, 6,90] 5
```

```
# assume user enter numbers with delimitor  " , "
x = [ int(x)  for x in input("Enter a list of ints: ").split(',')]
print (x, len(x))

Enter a list of ints:   12, 24,    1, 6,   90
 [12, 24, 1, 6,90] 5
```

- More about raw_input : https://docs.python.org/2/library/functions.html#raw_input

IMPORTANT Reference to the full list of Built-in functions:

- https://docs.python.org/2/library/functions.html#built-in-functions
- https://docs.python.org/3/library/functions.html

PYTHON DATA TYPES

Very important to keep in mind:

- Everything is object
- Think about each type as an own type of data structure
- Think about how the data is being stored.

Native data types –

- Booleans: True or False.
- Numbers:
- integers
- floats
- fractions, e.g.1/2 and 2/3
- complex numbers, e.g. 1j
- Strings: sequences of Unicode characters.
- Bytes and byte arrays
- Lists: ordered, mutable sequences of values.
- Tuples: ordered, immutable sequences of values.
- Dictionaries: are unordered segment of key-value pairs.

More elaborated types:

- Modules
- function,
- class
- method
- file
- even compiled C/C++ code.

Reference:

1) About Data Structure used in Python

https://docs.python.org/3/tutorial/datastructures.html

2) All intrinsic standard data types for version 3.6+:
https://docs.python.org/3/library/datatypes.html

3) All intrinsic data types for version 2.7.*:
https://docs.python.org/2/library/datatypes.html

4) Contain anything about Python from 2.x to the latest:

https://www.python.org/doc/

: You will even find their lexical table from this link - https://docs.python.org/3/reference/lexical_analysis.html; in case, someday you may want to create your own language.

All Python-3 Functions Reference:

https://docs.python.org/3/library/functions.html

SEQUENCE TYPES

As far as up to version 2.7, there are seven sequence types. You should always refer to the official document for the proper version to see the full list, and the latest update.

e.g. https://docs.python.org/2/library/stdtypes.html#sequence-types-str-unicode-list-tuple-bytearray-buffer-xrange.

The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

Basic mechanic : (https://docs.python.org/2/tutorial/introduction.html)

- Indexing, 0 means 1st element. Can use negative, that means from the end which starts with -1.
- Slicing, such as getting substring, sub-elements, etc.
- Matrixes, ie. array operations.

Only the following will be discussed in this document: str, list, tuple and bytearray.

with string with " "

```
str = 'Hello World!'
print str           # Hello World!
print str[0]        # H
print str[2:5]      # llo
print str[2:]       # llo World!
print str[-1]       # '!', i.e. refer to the character from the end.
print str[-6:-2]    # 'World'
print str[:-8]      # 'Hello'
print str * 2       # Hello World!Hello World!
print str + "!!"    # Hello World!!!
```

with List with [ ]

A list contains an ordered collection of mutable objects.

A single object is almost like a single node of C's struct with various data types.

Delete and Insert operations are expensive, as the list re-orders itself.

See online document for all Built-in List Functions and Methods.

```
#!/usr/bin/python

list1 = [ 'Curly', 100 , 3.1415, 'Larry', 1.61803 ]
tinylist = [123, 'Larry']
tinylist[1] = "Moe"     #
print (list1, len(list1))         # [ 'Curly', 100, 3.1415, 'Larry', 1.61803] 5
print (list1 + tinylist, len(list1))
                              #[ 'Curly', 100, 3.1415, 'Larry', 1.61803, 123, 'Moe'] 5
del  list1[2]
print (list1, len(list1))  #['Curly', 100, 'Larry', 1.61803] 4

print 3 in list1                  #    1.61803
for x in list1 :
      print x             #     Curly 100  3.1415  1.61803
tinylist.insert(1, 'and')
print tinylist            #   [123, 'and',  'Larry']
#-------------
a = [ 1, 2, 3, 4 ]
b = a           # like pointer in C, it is a reference;
print b                   # [1, 2, 3,4]
del a[1]
print a                          #error : name 'a' is not defined
#-------------
alist1   = [ 123, 'Larry' ]
alist2   = [ 456, 'Moe' ]
for i in alist1:
   print (i)                             #  123
                                         #  Larry
for i in alist1+ alist2:
   print (i, end='---')           #  123---Larry---456---Moe
#-------------
```

Slicing operator: *[ ] and [:]*

```
list = [ 'Moe', 123 , 3.5, 'Larry', "Curly" ]
 print (list[1:3])
 print (list[2:])
 print (list[:4])
```

Split the string into a list of float

```
x = "10,30,69,70"
print (x, "type: ", type(x), "len: ", len(x), "siz: ", sys.getsizeof(x))

n1 = list(map(float, x.split(',')))
print (n1, "type: ", type(n1), "len: ", len(n1), "siz: ", sys.getsizeof(n1))

      #---------------------
      # or you can do it in one line
n1 = [ int(x)  for x in input("Enter a list of ints, e.g. 1,3,4 : ").split(',')]
print (n1, "type: ", type(n1), "len: ", len(n1), "siz: ", sys.getsizeof(n1))
```

with Tuples with ( )

Tuples are similar to lists, except they are immutable. You can have nested tuples.

```
tup_long       = ( 'Curly', 100 , 3.1415, 'Larry', 1.61803 )
tup_short      = (123, 'Larry')
tup_str = “ice”,  “cream”
tup_nested     =  (123, 'Larry',  (456, ‘Benjamin’) )
print  tup_long[0]      # ‘Curly’
print tup_long[1:4]     #100, 3.1415, ‘Larry’, 1.61803
print 3 in tup_long     # ‘Larry’
for x in  tup_str:
    print  x, “-“       # ice-cream-
tup_new = tup_long + tup_short
del tup_long; del tup_short
```

Note the last example cause tup_long and tup_short being removed; like an object being deleted, or memory pointer being freed.

Being immutable, the following samples are invalid:

e.g.

```
tuple   =  ( 'EV3', 359, 3.1415, 'Larry', 1.61803 )
list    =  [ 'NXT', 299 , 3.23, 'Larry', 70.2  ]
tuple[2]       = 1000    # Invalid syntax with tuple
list[2]  = 1000    # Valid syntax with list
```

Common operations:

```
cmp(tuple1, tuple2)    len(tuple)     max(tuple)      min(tuple)
```

tuple(list) : convert a list to a tuple

See online document for all Built-in Tuples Functions

BYTEARRAY

Bytearray objects are created with the built-in function bytearray().

The bytearray class is a mutable sequence of integers in the range 0 <= x < 255.

Sample:

```
def doByteArray():
   elements = [100, 2, 5, 50, 255]
   values = bytearray(elements)    # mutable
  # if you use bytes(elements) , values[ ] will become immutable
   values[0] = 5
   values[1] = 0
   print "There are ", len(elements), "elements: " ,
   for value in values:
        print (value),
   return
```

Output : There are 5 elements: 5 0 5 50 255

Note:

If you are interesting to view the memory address, look into using class memoryview. E.g.

```
  v = memoryview('0123456789')
  print v[0:1]            # this syntax will return the starting address of the “v”.
```

LIST VS. TUPLES VS ARRAY

List [ ] AND [:]	Tuple ( )	Dictionary {key:value, … }	Array (from numpy module)
Mutable	Immutable	Mutable	Mutable
Reference by index	Reference by index	Reference by “key”. Key is an object	
Can hold heterogeneous data type	Can hold heterogeneous data type	--	Can hold only homogeneous data type
Can change such as append, extend, remove, etc.	Cannot	Can change such as append, extend, remove, etc.	Can change such as append, extend, remove, etc.
Can “find”	Same	same	same
Use the “in” operator	same	same	same
slower	faster	like list	fastest
Cannot be used as dictionary keys	Can be used as dictionary keys	--	Cannot be used as dictionary keys
ordered collection	ordered collection	No order	have functions that you can perform to them: e.g. x = array(‘i’, [3, 6, 9, 12]) x/3.0 x becomes [1, 2, 3, 4]

Stop here.

Should do some exercises on Python before proceeding.

MAPPING TYPES

with dictionary with { 'key1' : 'value', 'key2' : i-value, etc }

Dictionary are similar to lists, except they are two parts with the following restrictions:

Key-value pair: where key must be unique.

Value: can be any Python object, either standard objects or user-defined objects

Key: immutable object.

```
dictVar= {'Name':'Newton', 'Age': 7, 'Gender': 'Unknown'}
print  dictVar['Name'], " is ", dictVar['Age'], "years old."  #Newton  is 7 years old.
print "...."
dictVar['Age'] = 8;
dictVar['School'] = "PieSchool";
print dictVar        # School : PieSchool
print "...."
del dictVar['Name']
print dictVar        # School : PieSchool
print "...."
dictVar.clear( )             # remove all entries, but reference to dictVar still exists
del dictVar    # delete reference to dictVar, i.e. can no longer use dictVar
```

Output:

```
Newton  is  7 years old.
....
{'School': 'PieSchool', 'Gender': 'Unknown', 'Age': 8, 'Name': 'Newton'}
....
{'School': 'PieSchool', 'Gender': 'Unknown', 'Age': 8}
....
```

```
#---------------------------------------------
def f1():
   print( "Do Func 1.\n")
```

```
def f2():
    print( "Do Func 2.\n")

def f3():
     print( "Do Func 3.\n")

# map the inputs to the function blocks
options = { 3 : f3,            # simulate switch/case
          1 : f1,
          2 : f2
}
options[3]()
```

Mix and match techniques

```
zip( ) – map the similar index of multiple containers
name = [ "Joe", "Ethan", "Ashley", "Astha" ]
age = [ 10, 15, 9, 11 ]
marks = [ 40, 50, 60, 70 ]

# nicely formatted display
for n, a in zip(name, age):
    print ("Name :  %-20s     Age : %d" %(n, a))

# using zip() to map values
# converting values to print as list

mapped = zip(name, age, marks)
mapped = list(mapped)
print ("list: ", mapped)
```

How about unzip them back :

```
x = [1, 2, 3]
y = [4, 5, 6]
print ("x: ", type(x), x)         # x:  <class 'list'> [1, 2, 3]
print ("y: ", type(y), y)         # y:  <class 'list'> [4, 5, 6]

zipped = zip(x, y)
zipped = list(zipped)
print (zipped)

#unzip back to  two separate tuples
x2, y2 = zip(*zip(x, y))
print ("x2: ",type(x2),  x2)     # x:  <class 'tuple'> (1, 2, 3)
print ("y2: ",type(y2), y2)      # y:  <class 'tuple'> (4, 5, 6)
```

Creating matrix

```
mx =  ['*' for col in range(6)]
print(array)                              #['*', '*', '*', '*', '*', '*']

array =  [ ['*' for col in range(3)] for row in range(4) ]
print(array)            #      [['*', '*', '*'], ['*', '*', '*'], ['*', '*', '*'], ['*', '*', '*']]
print("\n".join(" ".join(row) for row in zip(*array)))  # nice 3x4 table of '*'
```

FLOW CONTROL WITH MAP AND LAMBDA (ADV. TOPICS)

map(function to apply, iterable list of inputs, ...)

- applies *function* to every item of *iterable*, yielding the results.
- If additional iterable arguments are passed, function must take that many arguments and is applied to the items from all iterables in parallel.
- With multiple iterables, the iterator stops when the shortest iterable is exhausted

e.g.

```
items = [1, 2, 3, 4, 5]
squared = []
for i in items:
   squared.append(i**2)
```

instead, do this:

```
items = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x**2, items))
```

Another sample:

```
def multiply(x):
   return (x*x)
def add(x):
   return (x+x)

funcs = [multiply, add]
for i in range(5):
   value = list(map(lambda x: x(i), funcs))
   print(value)

# Output:
# [0, 0]
# [1, 2]
# [4, 4]
# [9, 6]
# [16, 8]
```

Applies a rolling computation to sequential pairs of values in a list

e.g.

```
result = 1
list = [1, 2, 3, 4]
```

```
for num in list:
        result = result * num
print (result)                    # product = 24
```

Instead, do this with functools:

```
from functools import reduce

result = reduce((lambda x, y: x * y), [1, 2, 3, 4])
print (result)
# Output: 24
```

The **functools** module provides tools for working with functions and other callable objects, to adapt or extend them for new purposes without completely rewriting them.

About Data type conversion

This is like data casting.

To convert between types, you simply use the type name as a function.

Function	Description
int(x [,base])	Converts x to an integer. base specifies the base if x is a string.
long(x [,base])	Converts x to a long integer. base specifies the base if x is a string.
float(x)	Converts x to a floating-point number.
complex(real [,imag])	Creates a complex number.
str(x)	Converts object x to a string representation.
repr(x)	Converts object x to an expression string.
eval(str)	Evaluates a string and returns an object.
tuple(s)	Converts s to a tuple.
list(s)	Converts s to a list.
set(s)	Converts s to a set. Deprecated after version 2.7
dict(d)	Creates a dictionary. d must be a sequence of (key,value) tuples.
frozenset(s)	Converts s to a frozen set.
chr(x)	Converts an integer to a character.
unichr(x)	Converts an integer to a Unicode character.
ord(x)	Converts a single character to its integer value.
hex(x)	Converts an integer to a hexadecimal string.
oct(x)	Converts an integer to an octal string.

DEEP VS SHALLOW COPY

Shallow copy : is like a reference pointer C/C++

Deep copy : copy the value, including internal objects.

```
def prtA(a, str):
    print ("List %s:"%str, a, \
           " \n- type  : ", type(a), \
           "mem: ", id(a))
    Return

a = (3,10,2)
prtA(a, "a")
b = a       # b and a reference the same memory addr (shallow)
prtA(b, "b")
c = np.copy(a)    # c reference different address, but with a full copy (deep)
c[0] = 50
prtA(a, "a")
prtA(c, "c")
```

COMMAND LINE PARSER

- module argparse for reading (parsing) "–option" value pairs on the command line.

import argparse as ap p = ap.ArgumentParser()	To import and create the object
Add arguement	
p.add_argument('-v', '--velocity', type=float, default=0.0, help='initial velocity',)	'–v' : option associates with an input parameter 'velocity' : a dictionary key to give you the value of the argument type : default to is string if not specified default: default to none if not specified. help : automatically allow an option -h or --help that prints a usage string for all the given options.
Note the difference: p.add_argument('-v', '—velocity' vs. p.add_argument('--v', '—velocity' vs. p.add_argument('--v', '—velocity'	'--velocity' : "velocity" becomes the dictionary key to give you the value of the argument '––v' : "v" becomes the dictionary key to give you the value of the argument WRONG!
More examples	

p.add_argument(
'-v', '—velocity' *etc.*
vargs = vars(p.parse_args())
print (vargs["velocity"]*2.0)
	Read the command line arguments and interpret them.
	>>> python test.py -v 7.50
	>>> 15.0

p.add_argument('--v', *etc.* arg = p.parse_args() print(arg.v * 2) varg = vars(arg) print (varg["v"]*2.0)	>>> python test.py --v 5.0 10.0 10.0
p.add_argument("Sqr", help="Sqr number" , type=int) args = p.parse_args() print (args.Sqr**2)	>>> python test.py 4 >>> 16

ADVANCED TOPIC IN CONTROL FLOW USING LAMDA FUNCTIONS

Reference: https://docs.python.org/3/tutorial/controlflow.html

To be updated…

THE NUMPY MODULE

Note: There are many documents online about Numpy module. Without simply repeat what everyone does, this document will serve a concise document for experienced C/C++ programmer, with more focus on usage for mainly computer vision, and data analysis.

NumPy stands for Numerical Python. It is one of the most commonly used scientific and computing extensions in Python. It provides you with a way to represent images as a multi-dimensional array.

Free Alternative to MatLab

- Along with packages like SciPy (Scientific Python) and Mat–plotlib (plotting library), they replace common usage of Matlab.
- Another software GNU Octave also provides very compatible alternative to the expensive MatLab.

Functionalities :

- N-dimensional array generic data objects
- Broadcasting functions into numpy arrays
- Integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Installation:

For Raspberry Pi:

- should have been installed by default.
- Otherwise, do : sudo apt-get install python-numpy

For Windows:

- Use the standard “pip3” installation application which comes with Python.
- Best is to install the whole SciPy stack: http://scipy.org/install.html
- Download the package from : http://www.lfd.uci.edu/~gohlke/pythonlibs/#numpy. Find the latest one such as: numpy-1.11.3+mkl-cp27-cp27m-win_amd64.whl
- cd scripts
- pip3 install numpy-1.11.3+mkl-cp27-cp27m-win_amd64.whl

ABOUT NUMPY.ARRAY CLASS

Reference: https://docs.scipy.org/doc/numpy-1.13.0/reference/index.html

Numpy is a highly optimized library for numerical operations.

It is a library which:

1. enables creation of multidimensional array objects - ndarray (N-dimensional array | contiguous memory)
2. offers a collection of routines for processing those arrays.

It gives a MATLAB-style syntax. All the OpenCV array structures are converted to-and-from Numpy arrays.

For those die-hard C/C++ developers, you may be interested in viewing how this implement these structure, see : https://docs.python.org/3/c-api/typeobj.html.

Scope of this section:

- show you how to create N-dimensions arrays objects
- fundamental routines to operate on these arrays
- covers the basic operations including slicing and indexing which are used in basic operations on image processing in OpenCV.

using numpy vs. object

import numpy as np obj = np.array([1,5,4,6,7])	
	comment
memoryview(np)	Wrong! Has no buffer
obj	Type = Object of numpy.ndarray
obj.array , obj.ndarray	Wrong! No such attrib because obj is an object of numpy array
obj.shape, np.shape(obj)	Give you the shape of the obj
obj.ndim, obj.size, obj.itemsize np.ndim(obj), np.size(obj), np.itemsize(obj)	# of array structure ???? Total # of elements Byte counts of element
obj.dtype	Type = numpy.dtype dict
obj.data	Type = buffer
memoryview(obj)	8-bytes Memory address

About the “type”:

Do note “type” is an object type which stores a large number of values, mostly C function pointers, each of which implements a small part of the type’s functionality.)

Basics in creating multi-dimensional objects using np.array

*** array is immutable, i.e. you cannot change, cannot delete, etc.

a = np.array([1,2,3]) print (a) # now, a is an object of ndarray	# [1 2 3]
a = np.array([[1, 2], [3, 4]]) print (a)	[[1 2], [3 4]]
bt = np.array([1,2,3]) print ("2: ", bt, np.size(bt), np.shape(bt))	[1 2 3] 3 (3,)
bt = np.array([[1,2,3], [4, 5, 6]]) print (bt, "\n\n", np.size(bt), np.shape(bt), np.ndim(bt))	[[1 2 3] [4 5 6]] 6 (2, 3) 2
bt = np.array([True, True, True], dtype=bool) print (bt)	[True True True]
a = np.array([[[1, 2], [3, 4]]]) print (a)	[[[1 2], [3 4]]]
a= np.full((2,3), True) print (a)	[True True True]
a = np.zeros((2, 3) print (a) ---------- print(a.reshape(1,6), np.ndim(a)) ---------- print(a.reshape(3,2))	[[0. 0. 0.] [0. 0. 0.]] ---------- [[0. 0. 0. 0. 0. 0.]] ---------- ([[0., 0.], [0., 0.], [0., 0.]])
bt = np.array([1,2,3,4,5,6]) print("b: ", bt, np.ndim(bt)) bt2 = bt.reshape(1,6) print("new b: ", bt2 , np.ndim(bt2))	b: [1 2 3 4 5 6] 1 new b: [[1 2 3 4 5 6]] 2
a = np.array([1,2,3,4]) print (a, "dim:", a.ndim, "shape: ", a.shape)	[1 2 3 4] dim: 1 shape: (4,) ----------

---------- a = a.reshape(1,4).T ---------- a = np.array([[1,3,5], [2,6,10], [3, 9, 15]]) print (a, "dim: " , a.ndim, "shape: ", a.shape)	[[1] [2] [3] [4]] dim: 2 shape: (4, 1) ---------- [[1 3 5] [2 6 10] [3 9 15]] dim: 2 shape: (3, 3)

NP.NDARRAY

- multi-dimensional
- numpy.array is a function that returns a numpy.ndarray.
- Arrays should be constructed using array, zeros or empty.
- ndarray(...) is a lower level class for instantiating an array.

Basics in creating ndarray objects using np.ndarray

bt = np.ndarray(shape=(2,2), dtype=float) print (bt) ---------- bt.fill(5.0) print (bt)	[[0.00000000e+000 8.31937121e-312] [5.19960701e-313 8.90098126e-307]] Basically, just garbage data. ---------- [[5. 5.] [5. 5.]]
obj = np.array([[1,2], [3,4], [5,6]], dtype=np.float) print(obj.ndim, obj.size, obj.itemsize)	2 6 8
obj = np.array([[1,2], [3,4], [5,6]], dtype=np.int8) print(obj.ndim, obj.size, obj.itemsize)	2 6 1
bt = np.ndarray(shape=(2,3), dtype=float) bt.fill(6.5) if isinstance(bt, np.ndarray): print (bt) ---------- if isinstance(bt, np.ndarray):	 [[6.5 6.5 6.5] [6.5 6.5 6.5]] ---------- Type Error

Create a structure data type

structType = np.dtype([('age', np.int8)]) a = np.array([(10), (20), (30)], dtype= structType) print (a)	[(10,) (20,) (30,)]
print (a['age'])	[10 20 30]

CREATE ARRAY WITH RANGES

a = np.arange(6).reshape((2, 3)) print (a)	[[0 1 2] [3 4 5]]
a = np.arange(6) print (a)	[0 1 2 3 4 5]
for x in range(3): x = x ** 2 print (x)	0 1 4
for x in np.arange(4, 10, 2): x = x ** 2 print (x)	16 36 64
a = np.arange(10) print (a[0:5]) print (a[-1])	[2 3 4] 9

Try this and observe the object's attributes

```
def prtNP(a, str):
    print ("NP: %s:"%str, a, " \n- itemsize: ", a.itemsize, \
           " \n-- shape: ", a.shape, "| size: ", a.size, "dim: ", a.ndim, \
           "| dtype: ", a.dtype, " \n-- type  : ", type(a), \
           "mem: ", id(a))
    return

a = np.arange (3,10,2)
prtNP(a, "test")
```

SLICING

- read the parameters as (start: stop : step)

x = np.arange(10) print (x) s = slice(0,8,2) print (x[s])	[0 1 2 3 4 5 6 7 8 9] [0 2 4 6]
a = np.arange(10) print (a[::-1]) ---------- ra = a[5:2:-1] print (ra)	[9 8 7 6 5 4 3 2 1 0] ---------- [5 4 3]
a = np.array([[1,2,3],[4,5,6],[7,8,9]]) print (a[...,1]) ---------- print (a[1,...]) ---------- print (a[...,1:])	[2 5 8] ---------- [4 5 6] ---------- [[2 3] [5 6] [8 9]]

# slicing 	
z = a[0:, 1:]
print (z)
	[[2 3]
	[5 6]
	[8 9]]

z = a[1:, 2:] print (z)	[[6] [9]]
a = np.array([[1, 2, 3],[4, 5, 6],[7, 8, 9]]) b=np.flip(a,1) # vertical flip ---------- b=np.flip(a,0) # horizontical flip	# does a vertical flip [[3 2 1] [6 5 4] [9 8 7]] ---------- [[7 8 9] [4 5 6] [1 2 3]]

INDEXING

- Allow you to access and modify the array by indexing or slicing.
- 3 types of indexing methods: field access, basic slicing and advanced indexing.
- Syntax: slice(start, stop, and step)
- Review the online reference.
- Do the exercise in your student packet.

Sample: a = np.array([[1,2,3],[4,5,6],[7,8,9]])

y = a[[0,1,0], [1,1,2]] print (y)	[2 5 3]
rows = np.array([[0,0],[2,2]]) cols = np.array([[0,2],[0,2]]) y = a[rows,cols] print ("\n", y)	[[1 3] [7 9]] # a [[[0,0], [2,2]], [[0,2], [0,2]]] # 0,0 == 1 0,2 == 3 2,0== 7 2,2 == 9
z = a[0:3, 1:2] print (z)	[[2] [5] [8]]
z = a[0:3, 1:3] print (z)	[[2 3] [5 6] [8 9]]

Simpler example of file I/O:

```
x = np.arange( 2,20,3,  dtype=np.int)
print(x)

x.tofile("test.out")
y = np.fromfile("test.out", dtype=np.int)
print (y)
```

All others

- **A good reference and samples:** From Tutorials point.
- **Array creation routines**

  : https://docs.scipy.org/doc/numpy-1.12.0/reference/routines.array-creation.html

Questions

eye, nonzero, transpose, where,

Manipulations

array_split, reshape, resize, ndarray.item, concatenate, diagonal, dsplit, repeat,

Ordering

argmax, argmin, argsort, max, min

Operations

ndarray.fill, sum, put

arange, array, ones, zeros, copy, empty, tofile (refer to sample in fromfile), fromfile,

Python3 Exercises

BASICS

1. Write a Python program to print the following string in a specific format (see the output).

Output :

Twinkle, twinkle, little star,
How I wonder what you are!
Up above the world so high,
Like a diamond in the sky.
Twinkle, twinkle, little star,
How I wonder what you are

2. Write a Python program to display the current date and time.

Output:

```
Current date and time :
2014-07-05 14:34:14
```

3. Write a Python program to calculate number of days between two dates.

Sample dates : (2014, 7, 2), (2014, 7, 11)
Expected output : 9 days

4. Write a program to generate 1st 1000 prime numbers (prefer using Sieve of Eratosthenes.

5. Write a Python program to check whether a specified value is contained in a group of values. Go to the editor

Test Data :
3 -> [1, 5, 8, 3] : True
-1 -> [1, 5, 8, 3] : False

6. Write a Python program that accepts an integer (n) and computes the value of n+nn+nnn. Go to the editor

Sample value of n is 5
Expected Result : 615

WARM-UP WITH EASIER ONES

For the following exercises, make them all functions.

1. Ask users to enter numbers into a List.
2. Perform sum(), multiply() with the items in a list entered by users. Your program should also ask user which operation they wish to run.
3. Return the largest and smallest number from a list.
4. Find the size of each object in the list, and the size of the whole list.
5. Return the count of words that are >2 in len and have the same first and last char .

   Sample List : ['abc', 'xyz', 'aba', '1221']
   Expected Sample Result : 2

6. Create a list, sorted in increasing order by the last element in each tuple from a given list of non-empty tuples.
   Sample List : [(2, 5), (1, 2), (4, 4), (2, 3), (2, 1)]
   Expected Result : [(2, 1), (1, 2), (2, 3), (4, 4), (2, 5)]
7. Remove duplicates from a list.
8. Clone or copy a list.
9. What happened to the list:

   ```
   lst = ["HELLO", 5, [10], True]
   print(lst)
   lst[2].append(50)
   print(lst)
   ```

10. Give an arbitrary list (you can create one in your program). Then, ask yourself to enter a letter or phrase. Your program should check if the user-input exist in a given list.
11. Find the list of words that are longer than n from a given list of words.
12. Function to read in two lists and returns True if they have at least one common member
13. Print a specified list after removing the 0th, 4th and 5th elements.

    Sample List : ['Red', 'Green', 'White', 'Black', 'Pink', 'Yellow']
    Expected Output : ['Green', 'White', 'Black']
14. Generate a 3*4*6 3D array whose each element is *.

15. Write a Program to print the numbers of a specified list after removing even numbers from it.

16. Combine two lists, but remove all duplicates.

17. Get the frequency of the elements in a list.

18. Split a list of a set of numbers into 3x4

19. Create a list by concatenating a given list which range goes from 1 to n.
    Sample list : ['p', 'q']
    n =5
    Sample Output : ['p1', 'q1', 'p2', 'q2', 'p3', 'q3', 'p4', 'q4', 'p5', 'q5']

MORE DIFFICULT ONES

20. Check whether a list contains a sublist.
21. Check whether two lists are a palindrome .
22. Split a list every Nth element.
    Sample list: ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n']
    Expected Output: [['a', 'd', 'g', 'j', 'm'], ['b', 'e', 'h', 'k', 'n'], ['c', 'f', 'i', 'l']]
23. Compute the similarity between two lists.
    Sample data: ["red", "orange", "green", "blue", "white"], ["black", "yellow", "green", "blue"]
    Expected Output: Color1-Color2: ['white', 'orange', 'red']
                      Color2-Color1: ['black', 'yellow']

24. Replace the last element in a list with another list.
Sample data : [1, 3, 5, 7, 9, 10], [2, 4, 6, 8]
Expected Output: [1, 3, 5, 7, 9, 2, 4, 6, 8]

25. Find the list in a list of lists whose sum of elements is the highest.
    Sample lists: [1,2,3], [4,5,6], [10,11,12], [7,8,9]
    Expected Output: [10, 11, 12]

MINI-PROJECTS

26. Compute all prime numbers up to specified n using Sieve of Eratosthenes method for computing primes upto a specified number.
27.
28. creates a list of words, and determine the number of sets of anagrams.

TUPLE

1. Create a tuple tup = ("HELLO", 5, [], True), unpack them back into different variables
2. Expand the [ ] to 50 of them.
3. create the colon of a tuple.
4. Find the repeated items of a tuple.
5. Convert a list to a tuple.
6. Remove an item from a tuple.
7. Slice a tuple.
8. Find the length of a tuple. Then, return Nth item of a tuple. It should also check it the tuple is empty also.
9. unzip a list of tuples into individual lists.
10. reverse a tuple.
11. Print a tuple with string formatting.
12. Replace last value of tuples in a list. Do both of the following input sample.

    Sample list: [(10, 20, 40), (40, 50, 60), (70, 80, 90)]
    Expected Output: [(10, 20, 100), (40, 50, 100), (70, 80, 100)]

    Sample data: [(), (), ('',), ('a', 'b'), ('a', 'b', 'c'), ('d')]
    Expected output: [('',), ('a', 'b'), ('a', 'b', 'c'), 'd']

13. Sort a tuple by its float element.
    Sample data: [('item1', '12.20'), ('item2', '15.10'), ('item3', '24.5')]
    Expected Output: [('item3', '24.5'), ('item2', '15.10'), ('item1', '12.20')]
14. Count the elements in a list until an element is a tuple.

Dictionary

1. Convert a tuple to a dictionary.

2. Get the depth of a dictionary.

3. Check if all dictionaries in a list are empty or not.

   Sample list : [{},{},{}]
   Return value : True
   Sample list : [{1,2},{},{}]
   Return value : False

4. Convert list to list of dictionaries.

   Sample lists: ["Black", "Red", "Maroon", "Yellow"], ["#000000", "#FF0000", "#800000", "#FFFF00"]
   Expected Output: [{'color_name': 'Black', 'color_code': '#000000'}, {'color_name': 'Red', 'color_code': '#FF0000'}, {'color_name': 'Maroon', 'color_code': '#800000'}, {'color_name': 'Yellow', 'color_code': '#FFFF00'}]

5. Sort a list of nested dictionaries.

6. Remove key values pairs from a list of dictionaries.

7. Find a tuple, the smallest second index value from a list of tuples.

8. Create a list of empty d