30/9/14

Python

Setting up Python.

→ download Python 3 from Python.org.

→ Install it.

→ Download Eclipse IDE any version.

→ Then open Eclipse goto.

Help → Eclipse Marketplace → Search for PyDev and install it.

→ Then goto

Eclipse → ~~Pr~~eference...

PyDev → Interpreters → Python Interpreter

→ Here on the right Click on "New", before doing this, open terminal

Enter "which Python3", Copy that path on the program and goto eclipse

→ In the Interpreter Name: [ ] → Python 3.4 (anything)

Interpreter Executable: [ ] → paste the path copied from terminal

Click "Ok", Then finish. Then Complete the setting by click on "Ok" on Preference panel.

creating a new Project

→ ~~right~~ change the ~~View~~ Perspective by doing the following.

~~window → Show view~~

window → open perspective → other → PyDev ↵

→ In Package Explorer, right Click in the blank space

New → Project ↵

PyDev → PyDev Project → Next ↵

Project Name : [ ] → any name

Grammar Version : [▽] → 3.0

Interpreter : [▽] → Python 3.4

→ Click "Finish"

→ Right Click on the project Created

New → File ↵

File Name : [ ] → AnyText.py

↑
Important

Then Click Finish.

→ A new text file opens, where you can type in your code.

To run a python Script

→ In the title bar Search for ▷

↑
Click on it

Select Python Run → Click OK.

~~30 days of~~ ~~Python~~

I Hello World!

-> see "1_HelloWorld.py"

Print ("Hello, World")

II Conditional

->see "2_Conditional.py"

III Loops

->see "3_Loops.py"

IV Functions

-> see "4_function" folder

V Object Oriented Programming.

->see "5_OOP" folder

-> see "5_2_~~Inheritance~~Polymorphism.py" -> 2.3 is also inheritance.

-> model view Controller -> see "5_3_MVC.py"

VI Error reporting

->see "6_Exceptions.py"

2/16/14

General Syntax

I Creating a main script

→ see "1_Syntax.py".

II Conditionals

→ Else if is spelled in python as "elif"

III functions

→ to define or create a function we use the keyword "def"

Ex: def hi():
    print ("hi")

IV Objects:

→ Any thing created in python is an object (same like java).

→ See "8_Objects.py"

3/10/2014 ⑧ ③

Variable, objects and Values.

Understand Variables and Objects in Python

→ Every value in Python is an object, every thing including functions, variables and codes as well.

→ Every object has an ID, Type and Value.

- ID is a unique identification number
- Type could be <class 'int'> i.e. int, str etc...
- Value is the assigned value to an object.

eg: >>> X = 10

>>> Type (x)

<class 'int'>

>>> Id (x)

123456 -- This is some unique numbe which is dynamic.

Distinguishing mutable and immutable objects

→ Objects in Python can be mutable and immutable ~~objects~~

→ mutable objects may change value, immutable objects may not

- Sometimes you may think that immutable objects are changing the values, but in reality they are not.
- This can be checked by id( ).
- List, string etc... in Python are immutable objects, it may appear to change its value but in reality it does not.

eg. in IDLE

-> we should understand that an object has a variable that is variable
an ~~object~~ is an refrence to object

Variable ——reference——> Object.

eg: in IDLE.

let us consider

```
>>> x = 10
>>> type(x)
    <class 'int'>
>>> ID(x)
    12345
```

now lets assign another object to the same variable 'x'

```
>>> x=20
>>> type(x)
    <class 'int'>
>>> id(x)
    67890
```

now when you assign the same number 'x=10' and see its id.

```
>>> x=10
>>> id(x)
    12345
```

which shows that each object has an unique identifier but not the variable.

-> so you are just refering the same variable to diff. object.

→ most fundamental types in Python are immutable.
  - numbers, strings, tuples.
→ mutable objects include:
  - lists, dictionaries, other objects defendy upon implementation.

Using Numbers

→ see "9_Numbers.py"
→ Three Dede are two diff types of num in Python
  class 'int'
  class 'float'

Using strings

→ see "10_String"
→ Its an immutable objects.

Aggregating values with lists and tuples.

→ see "11_ListTuple"
→ Tuple is an immutable object, it cannot change its value
→ List is a mutable object which can change its value.

Creating associative lists with dictionaries:

→ see "12_Dictionary.py"
→ also called in general java associative array.
→ It is a mutable object.

Conditional

Conditional if and else:

Syntax: if ____:
        ____
        elif ____:
        ____
        else:
        ____

→ see " 13_ConditionalIfElse.py"

Switch:

→ see " 14_switch.py"

→ Python does not support switch statements instead we ~~for~~ use dictionary to select an object.

Loops

while Loop

Syntax: while ____:
        ____

→ see " 15_1_WhileLoop.py"

For Loop

→ see " 15_2_ForLoop.py.

6/10/2021

Syntax for ____ in ____ ← Container
        ____

any input given to it is a container object.

Enumerating iterations.

→ ~~The Enum~~ see " 15_3_Enumerating.py"

→ The Enumerate() method gives out the index value of each container object.

Syntax. enumerate(____)

→ This is usually used in loops.

loopControl

→ ~~(1) statary~~

→ see " 15_4_loopControl.py "

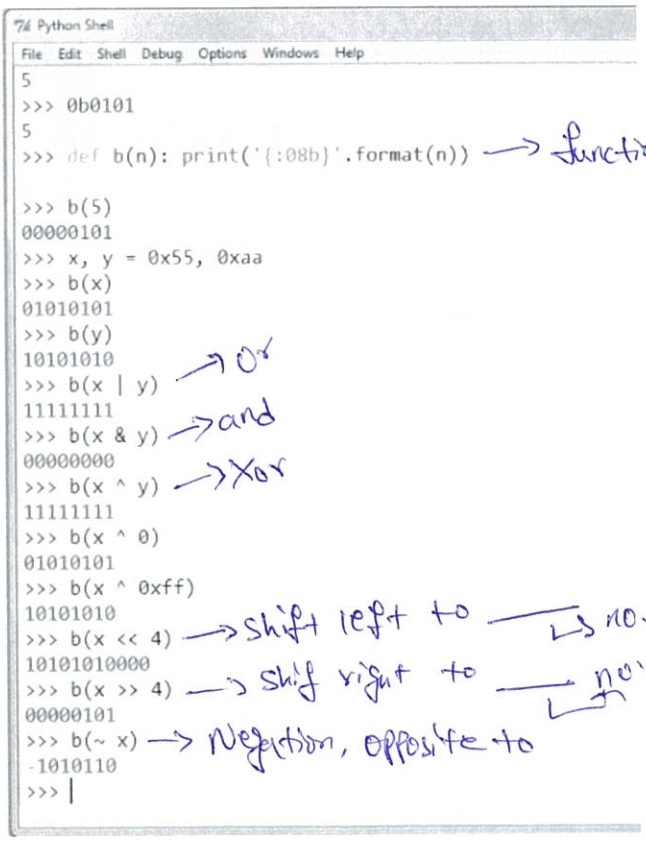
- Continue
- break
- else.

Operators

Operating on bitwise values

→

```
def b(n):
    print('{:08b}'.format(n))
```



Regular Expressions

-> used to Matching patterns in text.

-> It's a small language itself. , It can be very simple or very complex.

-> It is implemented in Python in "re" Module.

Searching

-> see "16_RegularExpression.py"

Syntax

```
import re
if re.search('_____', _____)
    Print(_____)
```

regular expression

input file o
some text

Replacing

-> see "17_RegularExpressionReplace.py"

Syntax

```
import re.
Print( re.sub('_____', '_____', _____) )
```

regular expression

replacement text

file or some text

Reusing

-> see "18_RegularExpressionReuse.py"

7/10/2014

Exceptions

Handeling Exceptions

-> see "19_1_Exception.py"

Syntax

```
try:
    ____          <- some code that may give out error.
except ____ as e:
    ____
else:
    ____          <- if you dont get the error Re do this.
```

Exception type

→ You can raise your own exception with "raise"

Raising an exception

→ see "19_2_RaiseException.py"

→ You can custom Eraise an exception with method "raise"

Syntax

```
if ____:
    ____
else:
    raise ____
```

condition

exception type to be raised if codition is not satisfied.

Functions

Defining functions:

→ see "20_1_Deffunction"

→ If you dont have anything in the function, just indent "pass"

Using List of arguments

→ see "20_2_List.py"

Named Parameters

→ see "20_3_NamedParameters.py"

Return Parameters

→ see "20_4_ReturnParameter.py"

Generator function

→ see "20_5_GenFunction.py"

→ An generator function is a function but returns an iterator object

→ also see "20_6_GenFunction.py".

Classes

- understanding classes and objects.

→ see "21_1_ClassesAndObjects.py"

8/10/2014

Method usage

→ see "21_2_Methods.py"

→ Lets say here is a class called Person(self, value), which takes a value.

→ Now we can use two instances say name1 and name2 to call it. This feature is called encapsulation.

Using Object Data

->see "21_3_ObjectData.py"

->Its a data which is carried around by the object.

->But when more number of setters and getters accumulate it is hard to keep a track of them so we use List (keyword arguments) **kwarg.

->see "21_4_ObjectData2.py"

Understand Inheritance

->see "21_5_Inheritance.py"

Understanding Polymorphism

->see "21_6_Polymorphism.py"

Using Generator

->see "21_7_@Generators.py"

-> it is an object that can be used in the context of an iterable, like in a for loop. see the example.

Using Decorators

->see "21_8_Decorators.py"

-> Decorators are special function which returns other function.

9/10/2014

String Methods

Understanding string as objects

→ A string like any other ~~object~~ type in Python 3 is an object,

→ A variable is only a reference to an object

→ so for ex!.

'This is a String' => This is an object of String.

s = 'This is a String' => r "also ... "

reference to string object.

So now as this is an object this can be also given as.

>>> 'This is a string'.upper() → upper case

THIS IS A STRING.

(or)

>>> s.upper() => will give this

So usually you will start seeing a string with out a reference with format() method.

>>> 'This is a string {}'.format(10)

This is a string 10.

Working with common Str methods

→ A string is an immutable object i.e. once assigned it cannot be changed.

→ So following methods are used in str/y

Capatilize() → capatilize the first str/y.

Upper() → converts to uppercase

lower() → lowercase (converts)

Swapcase() → swaps cases.

find('_') → finds the word and gives the address of it.

replace('_' (first to), '_' (that))

strip() → removes spaces at both ends of the str/y

rstrip() → removes space at end of the str/y.

rstrip('_')

↑ specify what to remove.

→ Boolean methods ~~expressions~~ in str/y are as follows:

isalnum() → checks if the str/y has only alphanumeric char

isalpha() → checks " " " " alpha "

isdigit() → " " " " digits "

isprintable() → " " " is printable or not.

Formatting strings with str.format

→ format returns a new string

→ eg:

```
>>> 'hello {}'.format('akshay')
hello akshay
```

→ There is something called as positional order in format() method.

Eg: let us consider

```
>>> a, b = (10, 20)
>>> 'This is {1}, This is also {1} but this is {0}'.format
(a, b)
This is 20, This is also 20 but this is 10
```

(a, b) → 0 1 → Just for reference.

You can specify the order at which the formatting should be done.

(Or)

You can also name them

eg: >>> This is {a1}, This is {b1}'.format(a1=a, b1=b)
This is 10, This is 20.

(Or)

You can also use dictionary style.

eg: >>> d = dict(a1=a, b1=b)
>>> 'This is {a1}, This is {b1}'.format(**d)
This is 10, This is 20.

Splitting and joining strings

→ The following methods are used for the same.

split() → splits each word in a string

eg: >>> S = 'This is a string'

>>> S.split()

['This', 'is', 'a', 'string']

>>> S.split('i') ⇒ splitting at 'i' position.

['Th', 'is ', 's a str', 'ng']

This is a white space not underscore

But we can still see the white spaces.

→ This string after splitting is iterable

eg: word = S.split()

>>> for i in word: print(i)

This
is
a
string.

→ you can join this but using join() method.

eg: >>> some_variable = ':'.join(word)

This:is:a:string

joins it with this.

Container

Creating sequence with tuples and lists.

tuples => are immutable (cannot be edited)

```
t = 1,2,3,4
    or
t = (1, 2, 3, 4)
    or
if you want to create one tuple.
t = (1,)
```

List => are Mutable (can be edited)

```
L = [1, 2, 3, 4]
    or
If you want to creat one list
L = [1,]
```

Common methods

```
min(_) → can be t or L → min num
max(_) → max num
_(-1) → last number
len(_) → length of _
type(_) → Type of _
```

to edit list

```
L[2] = 42 → address
[1, 2, 42, 4]
```

Operating of sequences with built in methods

→ other methods that can be used with Tuple and list are as follows.

only list	Both	only list
append(_) → insert anything at last	count(_)	remove(value) => removes the value in the list
extend(_) → insert multiple thing at last	index(_)	del L[12] → delete an item at address 12
insert(address, value) → insert at given address.		pop(_) → removes an item at given add. if not add given removes item from last.

Organising data with dictionaries

→ This is also known as associative arrays in java, You can do them by following ways.

```
>>> dict = {'one'=1, 'two'=2, 'three'=3}
```

(Or)

doing it with Constructor

```
d = dict(one=1, two=2, three=3)
```

→ Combining two dictionaries

eg: lets say

```
>>> x = dict(four=4, five=5, six=6)
>>> d = dict(one=1, two=2, three=3, **x)
```

The o/p is combine ~~and~~ d and x.

→ iterating through dictionaries are as follows.

```
for k in d: print(k)  => gives out keys.
for k, v in d.items(): print(k, v) => give key and values
```

→ To get a value

```
>>> d['one']
```

```
>>> d.get('one')
```

```
>>> d['four']
Exception.
```

```
>>> d.get('four')
```

give out nothing.

To over come Exception we can use get() method.

get(key, if nothing)

```
>>> d.get('four', 'not found')
not found
```

→ to delete.

del d['one']

(or)

d.pop('one') → pop give out the value but deletes the value in the dictionary.

Operating on Character data with bytes and byte arrays

→ see " 22_BytesAndByteArrays.py"

xml entity.

→ bytearray is mutable bytes not objects.

Files

Opening files

→ see folder 23_files in it.

"23_1_Opening files.py"

→ open(,) This is only an example it has many ~~options~~ options

input file name — mode type.

mode types =>
- r → read
- w → write
- a → append
- r+ → read and write
- rt → read text
- rb → read binary

writing files

→ see "23_2_ReadingFiles.py"

→ This example, reads every line in the text file and then writes it to the new file only after reading it. what if there is a big file with thousand lines

→ see "23_3_BufferWrite.py"

→ This can be done through buffer i/o mode.

→ alternat way of reading file is by following

eg: fh = ('text.txt', 'r')

print (fh.read())

→ Alternat way of write file.

fin = ('text.txt', 'r')

fout = ('new.txt', 'w')

someText = fin.read()

fout.write(someText)

Reading and writing Binary files

→ see "23_4_ReadWriteBinary.py"

an option to use is 'rb' and 'wb'

↑ read binary

↑ read binary.

Databases

Creating database with SQLite3

→ see "24_1_CreatingSQLite3.py"

→ Sqlite is a transactional db.

→ The db outputs a tuple which is a immutable object.

→ by putting an extra line ie.

```
db.row_factory = sqlite3.Row
```

makes all table an objects.

→ now these objects can be converted to dictionaries by using dict, or for more specific method, we can use

```
—['—']
```

(row name)

CRUD (see "24_2_CRUD)

↓

"sqlite3-crud.py" → see printed

Creating Database objects

see → 24_2_CRUD → "sqlite3-class.py"

→ see printed.

24_2CRUD → sqlite3-crud.py (12)

File - /Users/gollahalli/PycharmProjects/Python3/test.py

```
1  import sqlite3
2
3  def insert(db, row):
4      db.execute('insert into test (t1, i1) values (?, ?)', (row['t1'], row['i1']))
5      db.commit()
6
7  def retrieve(db, t1):
8      cursor = db.execute('select * from test where t1 = ?', (t1,))
9      return cursor.fetchone() # this will fetch one result
10
11 def update(db, row):
12     db.execute('update test set i1 = ? where t1 = ?', (row['i1'], row['t1']))
13     db.commit()
14
15 def delete(db, t1):
16     db.execute('delete from test where t1 = ?', (t1,))
17     db.commit()
18
19 def disp_rows(db):
20     cursor = db.execute('select * from test order by t1')
21     for row in cursor:
22         print('  {}: {}'.format(row['t1'], row['i1']))
23
24 def main():
25     db = sqlite3.connect('test.db')
26     db.row_factory = sqlite3.Row
27     print('Create table test')
28
29     db.execute('drop table if exists test')
30     db.execute('create table test ( t1 text, i1 int )')
31
32     print('Create rows')
33     insert(db, dict(t1 = 'one', i1 = 1))
34     insert(db, dict(t1 = 'two', i1 = 2))
35     insert(db, dict(t1 = 'three', i1 = 3))
36     insert(db, dict(t1 = 'four', i1 = 4))
37     disp_rows(db)
38
39     print('Retrieve rows')
40     print(dict(retrieve(db, 'one')), dict(retrieve(db, 'two')))
41
42     print('Update rows')
43     update(db, dict(t1 = 'one', i1 = 101))
44     update(db, dict(t1 = 'three', i1 = 103))
45     disp_rows(db)
46
47     print('Delete rows')
48     delete(db, 'one')
49     delete(db, 'three')
50     disp_rows(db)
51
52 if __name__ == "__main__": main()
53
```

#also Create

↳ dictionary type.

→ connecting db, if doesnot exist create it

24-2_CRUD → Sqlite3_Class.py (13)

File - /Users/gollahalli/PycharmProjects/Python3/test.py

```
1  import sqlite3
2
3  class database:
4      def __init__(self, **kwargs):
5          self.filename = kwargs.get('filename')
6          self.table = kwargs.get('table', 'test')
7
8      def sql_do(self, sql, *params):
9          self._db.execute(sql, params)
10         self._db.commit()
11             ↳local variables
12     def insert(self, row):
13         self._db.execute('insert into {} (t1, i1) values (?, ?)'.format(self._table), (row['t1'],
   row['i1']))
14         self._db.commit()
15
16     def retrieve(self, key):
17         cursor = self._db.execute('select * from {} where t1 = ?'.format(self._table), (key,)
   )
18         return dict(cursor.fetchone())
19
20     def update(self, row):
21         self._db.execute(
22             'update {} set i1 = ? where t1 = ?'.format(self._table),
23             (row['i1'], row['t1']))
24         self._db.commit()
25
26     def delete(self, key):
27         self._db.execute('delete from {} where t1 = ?'.format(self._table), (key,))
28         self._db.commit()
29
30     def disp_rows(self):
31         cursor = self._db.execute('select * from {} order by t1'.format(self._table))
32         for row in cursor:
33             print('  {}: {}'.format(row['t1'], row['i1']))
34
35     def __iter__(self):
36         cursor = self._db.execute('select * from {} order by t1'.format(self._table))
37         for row in cursor:
38             yield dict(row)
39
40     @property
41     def filename(self): return self._filename
42
43     @filename.setter
44     def filename(self, fn):
45         self._filename = fn
46         self._db = sqlite3.connect(fn)
47         self._db.row_factory = sqlite3.Row
48
49     @filename.deleter
50     def filename(self): self.close()
51
52     @property
53     def table(self): return self._table
54     @table.setter
55     def table(self, t): self._table = t
56     @table.deleter
57     def table(self): self._table = 'test'
```

takes in dictionary.
instance of db from main.
→gives only one o/p.

```
58
59      def close(self):
60          self._db.close()
61          del self._filename
62
63  def main():
64      db = database(filename = 'test.db', table = 'test')
65
66      print('Create table test')
67      db.sql_do('drop table if exists test')
68      db.sql_do('create table test ( t1 text, i1 int )')
69
70      print('Create rows')
71      db.insert(dict(t1 = 'one', i1 = 1))
72      db.insert(dict(t1 = 'two', i1 = 2))
73      db.insert(dict(t1 = 'three', i1 = 3))
74      db.insert(dict(t1 = 'four', i1 = 4))
75      for row in db: print(row)
76
77      print('Retrieve rows')
78      print(db.retrieve('one'), db.retrieve('two'))
79
80      print('Update rows')
81      db.update(dict(t1 = 'one', i1 = 101))
82      db.update(dict(t1 = 'three', i1 = 103))
83      for row in db: print(row)
84
85      print('Delete rows')
86      db.delete('one')
87      db.delete('three')
88      for row in db: print(row)
89
90  if __name__ == "__main__": main()
91
```

10/10/2014

Modules (25_modules)

Using standard library Modules.

→ see "25_1_LibModules.py"

Finding third-party modules:

→ You can find 3rd party modules at

PyPi.Python.org/pypi

→ installing a Python3 model do the following.

Python3 _____ install → This should be done in terminal.

↑
Module name
(name it will
be setup.py or install.py
this file can be found in the
Model folder after downloading

→ see "25_2_3rdModule.py" for an example

Creating a module:

→ see "25_3_CreateModule"

13/10/14

Building a database application

Normalizing a database interface

See "26_1_NormalizingDB.py"

-> See more example file for references.