1. Slot

\_\_slot\_\_\_ use in coding will reduce ram load

class MyClass(object):

\_\_slots\_\_ = ['name', 'identifier']

def \_\_init\_\_(self, name, identifier):

self.name = name

self.identifier = identifier

self.set\_up()

1. Mutation: Able to change
2. Enumerate counter
3. Object introspection (Dir, type, Id, inspect module)
4. Comprehensions (list, dictionary, set, generator)

list -

a\_list = [1, ‘4’, 9, ‘a’, 0, 4]

squared\_ints = [ e\*\*2 for e in a\_list if type(e) == types.IntType ]

print squared\_ints

# [ 1, 81, 0, 16 ]

dictionary-

mcase = {'a':10, 'b': 34, 'A': 7, 'Z':3}

mcase\_frequency = { k.lower() : mcase.get(k.lower(), 0) + mcase.get(k.upper(), 0) for k in mcase.keys() }

# mcase\_frequency == {'a': 17, 'z': 3, 'b': 34}

set-

names = [ 'Bob', 'JOHN', 'alice', 'bob', 'ALICE', 'J', 'Bob' ]

{ name[0].upper() + name[1:].lower() for name in names if len(name) > 1 }

We require the set-{ 'Bob', 'John', 'Alice' }

Generator-

1. exception (try/except, multiple exception, finally, try/else)
2. Lambdas
3. One-liners
4. for/else
5. open function
6. introspection

Dir, type, id, inspect module

1. concurrency
2. distributed system
3. coroutines
   1. generators are data producers
   2. coroutines are data consumers
4. function caching

from functools import lru\_cache

@lru\_cache(maxsize=32)

def fib(n):

if n < 2:

return n

return fib(n-1) + fib(n-2)

>>> print([fib(n) for n in range(10)])

# Output: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

1. context manager(with,)

The with statement stores the \_\_exit\_\_ method of the File class.

It calls the \_\_enter\_\_ method of the File class.

The \_\_enter\_\_ method opens the file and returns it.

The opened file handle is passed to opened\_file.

We write to the file using .write().

The with statement calls the stored \_\_exit\_\_ method.

The \_\_exit\_\_ method closes the file.

It passes the type, value and traceback of the error to the \_\_exit\_\_ method.

It allows the \_\_exit\_\_ method to handle the exception.

If \_\_exit\_\_ returns True then the exception was gracefully handled.

If anything, other than True is returned by the \_\_exit\_\_ method then the exception is raised by the with statement.

1. python c extension
2. targeting py 2+3
3. \*args and \*\*kwargs
4. Set
5. decorators
6. function Decorators
   * + 1. nested function
       2. function return function
       3. reference function
       4. function as parameters
7. Class Decorators

1. Debugging

i)python -m pdb my\_script.py

**ii)import** **pdb**

**def** make\_bread():

pdb.set\_trace()

**return** "I don't have time"

print(make\_bread())

iii)Dis

iv) Tabnanny

v) cprofile

1. venv
2. generators

**1)def** generator\_function():

**for** i **in** range(10):

**yield** i

**for** item **in** generator\_function():

print(item)

2) **def** fibon(n):

a = b = 1

**for** i **in** range(n):

**yield** a

a, b = b, a + b

**for** x **in** fibon(1000000):

print(x)

3) **def** fibon(n):

a = b = 1

result = []

**for** i **in** range(n):

result.append(a)

a, b = b, a + b

**return** result

4)

1. collection (defaultdict, OrderedDict, counter, deque, namedtuple, enum.Enum)
2. maps, filter, reduce
3. formatting
4. Ternary operator
5. Dictionary, list, tuple, string, array
6. PEP8
7. bug finder- PyChecker, Pylint
8. namespace
9. docstring
10. #!/usr/local/bin/python
11. You can access a module written in Python from C by following method,

Module = =PyImport\_ImportModule("<modulename>");

1. files I/O
2. unittest
3. Dogpile
4. Memcached
5. XML, Csv, Pdf, Json
6. Monkey patch
7. Regular expression
8. SORT
9. OOPS
10. Async, await, promise
11. CGI Programming
12. Constructor, destructor, abstract, static, dynamic binding, operator overloading
13. Pickling and Unpicking