readline and readlines functions

Ethans
Learn from experts

```
>>> help(infile.readline)
readline(...)
   readline([size]) -> next line from the file, as a string.

   Retain newline.  A non-negative size argument limits the maximum
   number of bytes to return (an incomplete line may be returned then).
   Return an empty string at EOF.

>>> help(infile.readlines)

readlines(...)
   readlines([size]) -> list of strings, each a line from the file.

   Call readline() repeatedly and return a list of the lines so read.
   The optional size argument, if given, is an approximate bound on the
   total number of bytes in the lines returned.
```


readline and readlines functions

Ethans
Learn from experts

```
>>> infile = open(r'C:\Users\jatin\Desktop\Ethans\test.txt', 'r')

>>> print infile.readline()
This is line number 1 in the file

>>> print infile.readlines()
['This is line number 2 in the file\n', 'This is line number 3 in the file']

>>> print infile.tell()
103

>>> infile.seek(0,0)

>>> print infile.readlines()
['This is line number 1 in the file\n', 'This is line number 2 in the file\n', 'This is line
number 3 in the file']
```

Write file

Ethans
Learn from experts

```
>>> outfile = open(r'C:\Users\jatin\Desktop\Ethans\test1.txt', 'w')
>>> outfile.closed
False
>>> dir(outfile)
['__class__', '__delattr__', '__doc__', '__enter__', '__exit__', '__format__',
'__getattribute__', '__hash__', '__init__', '__iter__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'close', 'closed', 'encoding', 'errors', 'fileno', 'flush', 'isatty', 'mode', 'name', 'newlines',
'next', 'read', 'readinto', 'readline', 'readlines', 'seek', 'softspace', 'tell', 'truncate',
'write', 'writelines', 'xreadlines']
```


Data descriptors

Ethans
Learn from experts

```
| closed
|     True if the file is closed
|
| encoding
|     file encoding
|
| errors
|     Unicode error handler
|
| mode
|     file mode ('r', 'U', 'w', 'a', possibly with 'b' or '+' added)
|
| name
|     file name
|
| newlines
|     end-of-line convention used in this file
```

write function

Ethans
Learn from experts

```
>>> outfile = open(r'C:\Users\jatin\Desktop\Ethans\test1.txt', 'w')
>>> outfile.write('This is line number 1 in the file\n')
>>> # Content is not yet written in the file, we need to flush or close object


>>> help(outfile.write)
Help on built-in function write:

write(...)
   write(str) -> None.  Write string str to file.

   Note that due to buffering, flush() or close() may be needed before
   the file on disk reflects the data written.
```


flush and close function

Ethans
Learn from experts

```
>>> help(outfile.flush)
Help on built-in function flush:

flush(...)
   flush() -> None.  Flush the internal I/O buffer.

>>> help(outfile.close)
Help on built-in function close:

close(...)
   close() -> None or (perhaps) an integer.  Close the file.

   Sets data attribute .closed to True.  A closed file cannot be used for
   further I/O operations.  close() may be called more than once without
   error.  Some kinds of file objects (for example, opened by popen())
   may return an exit status upon closing.
```

writelines function

Ethans
Learn From experts

```
>>> help(outfile.writelines)
Help on built-in function writelines:

writelines(...)
  writelines(sequence_of_strings) -> None.  Write the strings to the file.

  Note that newlines are not added.  The sequence can be any iterable object
  producing strings. This is equivalent to calling write() for each string.
```

```
>>> infile = open(r'C:\Users\jatin\Desktop\Ethans\test.txt', 'r')
>>> outfile = open(r'C:\Users\jatin\Desktop\Ethans\test1.txt', 'w')
>>> outfile.write(infile.readlines())
>>> outfile.close()
```


os module

Ethans
Learn From experts

This module provides a portable way of using operating system dependent functionality

```
>>> import os
>>> dir(os)
['abort', 'access', 'altsep', 'chdir', 'chmod', 'close', 'closerange', 'curdir', 'defpath', 'devnull',
'dup', 'dup2', 'environ', 'errno', 'error', 'execl', 'execle', 'execlp', 'execlpe', 'execv', 'execve',
'execvp', 'execvpe', 'extsep', 'fdopen', 'fstat', 'fsync', 'getcwd', 'getcwdu', 'getenv', 'getpid',
'isatty', 'kill', 'linesep', 'listdir', 'lseek', 'lstat', 'makedirs', 'mkdir', 'name', 'open', 'pardir',
'path', 'pathsep', 'pipe', 'popen', 'popen2', 'popen3', 'popen4', 'putenv', 'read', 'remove',
'removedirs', 'rename', 'renames', 'rmdir', 'sep', 'spawnl', 'spawnle', 'spawnv', 'spawnve',
'startfile', 'stat', 'stat_float_times', 'stat_result', 'statvfs_result', 'strerror', 'sys', 'system',
'tempnam', 'times', 'tmpfile', 'tmpnam', 'umask', 'unlink', 'unsetenv', 'urandom', 'utime',
'waitpid', 'walk', 'write']
```

https://docs.python.org/2/library/os.html

os module functions help

Ethans
Learn from experts

```
>>> help(os.getenv)
Help on function getenv in module os:

getenv(key, default=None)
    Get an environment variable, return None if it doesn't exist.
    The optional second argument can specify an alternate default.
```


os module functions

Ethans
Learn from experts

```
>>> os.getenv('OS')
'Windows_NT'

>>> os.getcwd()
'C:\\Python27'

>>> os.getcwdu()
u'C:\\Python27'

>>> os.getpid()
10124

>>> os.chdir('..')
>>> os.getcwd()
'C:\\'

>>> os.listdir('.')
['Anaconda2', 'Ethans', 'Perl64', 'Program Files', 'Program Files (x86)', Python27', 'sqlite',
'swapfile.sys', 'ubuntu', 'Users', 'Windows', 'Windows.old', 'wubildr', 'wubildr.mbr']
```

os module functions

Ethans
Learn From experts

```
>>> os.chdir('C:\Users\jatin\Desktop\Ethans')

>>> os.getcwd()
'C:\\Users\\jatin\\Desktop\\Ethans'

>>> os.mkdir('DemoDir')

>>> os.remove('file.txt')

>>> os.stat('test.txt')
nt.stat_result(st_mode=33206, st_ino=0L, st_dev=0, st_nlink=0, st_uid=0, st_gid=0,
st_size=103L, st_atime=1470715035L, st_mtime=1470715057L, st_ctime=1470715035L)

>>> os.stat('test.txt')[6]
103L

>>> os.system('cls')
>>> file = os.popen('dir')
>>> file.read() # return all data of dir command
```


os.path module functions

Ethans
Learn From experts

This module implements some useful functions on pathnames. To read or write files see open(), and for accessing the filesystem see the os module.

```
>>> dir(os.path)
['__all__', '__builtins__', '__doc__', '__file__', '__name__', '__package__',
'_abspath_split', '_getfullpathname', 'abspath', 'altsep', 'basename', 'commonprefix',
'curdir', 'defpath', 'devnull', 'dirname', 'exists', 'expanduser', 'expandvars', 'extsep',
'genericpath', 'getatime', 'getctime', 'getmtime', 'getsize', 'isabs', 'isdir', 'isfile', 'islink',
'ismount', 'join', 'lexists', 'normcase', 'normpath', 'os', 'pardir', 'pathsep', 'realpath',
'relpath', 'sep', 'split', 'splitdrive', 'splitext', 'splitunc', 'stat', 'supports_unicode_filenames',
'sys', 'walk', 'warnings']
```

os.path module functions

Ethans
Learn from experts

```
>>> os.path.isdir('DemoDir')
True
>>> os.path.isfile('test.txt')
True
>>> os.path.getsize('test.txt')
103L
>>> os.path.exists('blabla')
False
>>> os.path.basename(r'C:\Users\jatin\Desktop\Ethans\test.txt')
'test.txt'
>>> os.path.getmtime('test.txt')
1470715057.379443
```


Traverse a Directory Tree in Python

Ethans
Learn from experts

```
import os
for dirName, subdirList, fileList in os.walk('.'):
    print 'Found directory: %s' % dirName
    for fname in fileList:
        print '\t%s' % fname

'''
dirName: The next directory it found.
subdirList: A list of sub-directories in the current directory.
fileList: A list of files in the current directory.
'''
```

Objective – Module 6

Ethans
Learn From experts

Python Modules and Packages

- Python inbuilt Modules
- os, sys, datetime, time, random, zip modules
- Create Python UDM – User Defined Modules
- Define PYTHONPATH
- Create Python Packages
- init File for package initialization

Slide 135

Inbuilt Modules

Ethans
Learn From experts

The Python Language Reference describes the exact syntax and semantics of the Python language, this library reference manual describes the standard library that is distributed with Python. It also describes some of the optional components that are commonly included in Python distributions.

Python's standard library is very extensive, offering a wide range of facilities as indicated by the long table of contents listed below. The library contains built-in modules (written in C) that provide access to system functionality such as file I/O that would otherwise be inaccessible to Python programmers, as well as modules written in Python that provide standardized solutions for many problems that occur in everyday programming. Some of these modules are explicitly designed to encourage and enhance the portability of Python programs by abstracting away platform-specifics into platform-neutral APIs.

More info: https://docs.python.org/2/library/

Slide 136

www.ethans.co.in

Common Modules

Ethans
Learn from experts

os — Miscellaneous operating system interfaces
os.path - Common pathname manipulations
pickle — Python object serialization
getopt — C-style parser for command line options
logging — Logging facility for Python
subprocess — Subprocess management
socket — Low-level networking interface
timeit — Measure execution time of small code snippets
datetime — Basic date and time types
math — Mathematical functions
shutil — High-level file operations
zipfile — Work with ZIP archives
sqlite3 — DB-API 2.0 interface for SQLite databases
re — Regular expression operations
smtplib — SMTP protocol client
smtpd — SMTP Server
telnetlib — Telnet client
More info: https://docs.python.org/2/library/

datetime

Ethans
Learn from experts

```
>>> import datetime

>>> dir(datetime)
['MAXYEAR', 'MINYEAR', '__doc__', '__name__', '__package__', 'date', 'datetime',
'datetime_CAPI', 'time', 'timedelta', 'tzinfo']

>>> dir(datetime.datetime)
['__add__', '__class__', '__delattr__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__gt__', '__hash__', '__init__', '__le__', '__lt__', '__ne__',
'__new__', '__radd__', '__reduce__', '__reduce_ex__', '__repr__', '__rsub__',
'__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', 'astimezone',
'combine', 'ctime', 'date', 'day', 'dst', 'fromordinal', 'fromtimestamp', 'hour',
'isocalendar', 'isoformat', 'isoweekday', 'max', 'microsecond', 'min', 'minute', 'month',
'now', 'replace', 'resolution', 'second', 'strftime', 'strptime', 'time', 'timetuple',
'timetz', 'today', 'toordinal', 'tzinfo', 'tzname', 'utcfromtimestamp', 'utcnow',
'utcoffset', 'utctimetuple', 'weekday', 'year']
```

datetime

Ethans
Learn from experts

```
# Construct date
>>> t = datetime.time(1, 2, 3) # Contruct a time
>>> t.hour # returns 1
>>> t.minute # returns 2

>>> d = datetime.datetime(2015, 6, 6)
>>> d.year # return 2015

# Date Arithmetic
>>> today = datetime.date.today()          # Print current server date
2016-08-11
>>> one_day = datetime.timedelta(days=1)
>>> print today + one_day

# Compare dates
>>> tomorrow = today + one_day
>>> tomorrow > today
True
```


datetime formatting

Ethans
Learn from experts

```
import datetime

format = "%a %b %d %H:%M:%S %Y"

today = datetime.datetime.today()
print 'ISO     :', today

s = today.strftime(format)
print 'strftime:', s

d = datetime.datetime.strptime(s, format)
print 'strptime:', d.strftime(format)
```

time module

Ethans
Learn from experts

```
# Construct time
>>> import time
>>> print 'The time is:', time.time()
The time is: 1470889543.66
>>> print 'The time is:', time.ctime()
The time is: Thu Aug 11 09:56:05 2016

>>> print 'gmtime  :', time.gmtime()
gmtime  : time.struct_time(tm_year=2016, tm_mon=8, tm_mday=11, tm_hour=4,
tm_min=29, tm_sec=41, tm_wday=3, tm_yday=224, tm_isdst=0)
>>> print 'localtime:', time.localtime()
localtime: time.struct_time(tm_year=2016, tm_mon=8, tm_mday=11, tm_hour=9,
tm_min=59, tm_sec=52, tm_wday=3, tm_yday=224, tm_isdst=0)

# Arithmetic on time
>>> print '15 minutes later: ', time.ctime(time.time() + 15)
15 minutes later:  Thu Aug 11 09:58:53 2016
```


time module

Ethans
Learn from experts

```
# Construct time
>>> import time
>>> print 'The time is:', time.time()
The time is: 1470889543.66
>>> print 'The time is:', time.ctime()
The time is: Thu Aug 11 09:56:05 2016

>>> print 'gmtime  :', time.gmtime()
gmtime  : time.struct_time(tm_year=2016, tm_mon=8, tm_mday=11, tm_hour=4,
tm_min=29, tm_sec=41, tm_wday=3, tm_yday=224, tm_isdst=0)
>>> print 'localtime:', time.localtime()
localtime: time.struct_time(tm_year=2016, tm_mon=8, tm_mday=11, tm_hour=9,
tm_min=59, tm_sec=52, tm_wday=3, tm_yday=224, tm_isdst=0)

# Arithmetic on time
>>> print '15 minutes later: ', time.ctime(time.time() + 15)
15 minutes later:  Thu Aug 11 09:58:53 2016
```


www.ethans.co.in

time formatting

Ethans
Learn from experts

```
import time

now = time.ctime()
print now

parsed = time.strptime(now)
print "%d-%d-%d" %(parsed[0],parsed[1], parsed[2])

print time.strftime("%a %b %d %H:%M:%S %Y", parsed)
```

```
Thu Aug 11 10:02:38 2016
2016-8-11
Thu Aug 11 10:02:38 2016
```


random module

Ethans
Learn from experts

random – Pseudorandom number generators

```
>>> import random
>>> random.random()
0.9047586792455821

>>> print random.randint(1, 100)
60
```

```
import random

with open('words.txt', 'r') as f:
   words = f.readlines()
   words = [ w.rstrip() for w in words ]

for w in random.sample(words, 5):
   print w
```

zip module

Ethans
Learn from experts

```
zf = zipfile.ZipFile('zipfile_write.zip')
for filename in [ 'LICENSE.txt']:
    data = zf.read(filename)
    print filename, ':'
    print repr(data)
```

```
import zipfile

print 'creating archive'
zf = zipfile.ZipFile('zipfile_write.zip', mode='w')
print 'adding README.txt'
zf.write('LICENSE.txt')
zf.close()

print zipfile.is_zipfile('zipfile_write.zip')
```


User Defined Module

Ethans
Learn from experts

- You may want to split your program into several files for easier maintenance. You may also want to use a handy function that you've written in several programs without copying its definition into each program.
- A module is a file containing Python definitions and statements. The file name is the module name with the suffix .py appended. Within a module, the module's name (as a string) is available as the value of the global variable __name__..
- A module can contain executable statements as well as function definitions

Syntax:

import module1[, module2[,... moduleN]

Locate module

Ethans
Learn from experts

There are (at least) three kinds of modules in Python:

modules written in Python (.py);
modules written in C and dynamically loaded (.dll, .pyd, .so, .sl, etc);
modules written in C and linked with the interpreter.

To find the location of the file:

>>> import math and >>> math.__file__ (Error as here no file is available)
>>> import os, os.__file__ # 'C:\\Python27\\lib\\os.pyc'
>>> import random, >>> random.__file__, usr/lib/python2.4/random.pyc'

Create Module

Ethans
Learn from experts

For example:

```
# Import module support
import support
# Now you can call defined function that module as follows
support.print_func("Zara")
```

Do following:

Create a simple script/program with .py extension.
Import that program to your namespace and called its object.
Write a program to create Fibonacci module and print the first 20 Fibonacci numbers.

Search Module

Ethans
Learn from experts

Python search the PYTHONPATH for the module folder to get them imported to the namespace. We will discuss in details in next slides.

For Now if we want to import any of the module. We append the sys.path by the path we required.

For Example:

```
>>> sys.path.append('C:/Users/Jatin/Desktop/Python Scripts')

>>> import studentMarks2
Jatin has average marks: 75
```


Module Symbol table

Ethans
Learn from experts

Each module has its own private symbol table, which is used as the global symbol table by all functions defined in the module.

You can also import the specific methods of modules as well.

Syntax:

- ***from module import*** *function1, function2*

Now you can call that function directly:
function1(500)

from module import *
Import all methods and variable except all stating with _ (Avoid it)

Standard Boiler Plate

Ethans
Learn from experts

When a Python file is run directly, the special variable "__name__" is set to "__main__". Therefore, it's common to have the boilerplate if __name__ ==... shown above to call a main() function when the module is run directly, but not when the module is imported by some other module.

```
#!/usr/bin/python

# import modules used here -- sys is a very standard one
import sys

# Gather our code in a main() function
def main():
    print 'Hello there', sys.argv[1]
    # Command line args are in sys.argv[1], sys.argv[2] ...
    # sys.argv[0] is the script name itself and can be ignored

# Standard boilerplate to call the main() function to begin
# the program.
if __name__ == '__main__':
    main()
```

Slide 151

www.ethans.co.in

Search Path

Ethans
Learn from experts

The interpreter first searches for a built-in module with that name.

The directory containing the input script (or the current directory).

PYTHONPATH (a list of directory names, with the same syntax as the shell variable PATH).

The installation-dependent default.

These search paths would be in a list of directories given by the variable sys.path

set PYTHONPATH=C:\\Users\\Sony\\ Desktop\\Python Batch 29 Oct;C:\Users\Sony\Desktop\Python Batch 29 Oct\Day 3

Slide 152

www.ethans.co.in

Python Package

Ethans
Learn from experts

What is package?

Packages are a way of structuring Python's module namespace.

How it works?

- Suppose you want to design a collection of modules (a "package") for the uniform handling of employee files and department data.
- When importing the package, Python searches through the directories on sys.path looking for the package subdirectory.
- The __init__.py files are required to make Python treat the directories as containing packages, this is done to prevent directories with a common name.

__init__.py

Ethans
Learn from experts

__init__.py is the mandatory file available in the directory to make it package

```
root\
    system1\
        __init__.py
        utilities.py
        main.py
        other.py
    system2\
        __init__.py
        utilities.py
        main.py
        other.py
    system3\                    # Here or elsewhere
        __init__.py             # Your new code here
        myfile.py
```

import everything

Ethans
Learn From experts

What happens when the user writes
from root.system1 import *?

The import statement uses the following convention: if a package's __init__.py code defines a list named __all__, it is taken to be the list of module names that should be imported when from package import * is encountered.

Usually __init__.py contains the following code:

```
__all__ = ["echo", "surround", "reverse"]
```


Some imp functions

Ethans
Learn From experts

➢ filter(function, sequence)
returns a sequence consisting of those items from the sequence for which function(item) is true.

If sequence is a string or tuple, the result will be of the same type; otherwise, it is always a list. For example, to compute primes up to 25:

```
>>> def f(x): return x % 2 != 0 and x % 3 != 0
...
>>> filter(f, range(2, 25))
[5, 7, 11, 13, 17, 19, 23]
```

➢ map(function, sequence)
calls function(item) for each of the sequence's items and returns a list of the return values. For example, to compute some cubes:

```
>>> def cube(x): return x*x*x
>>> map(cube, range(1, 11))
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

Objective – Module 7

Ethan's Learn from experts

Exception Handling

- Python Exceptions Handling
- What is Exception?
- Handling various exceptions using try....except...else
- Try-finally clause
- Argument of an Exception and create self exception class
- Python Standard Exceptions
- Raising an exceptions, UD Exceptions

OOPS Python

- Object oriented features
- Understand real world examples on OOP
- Implement Object oriented with Python
- Creating Classes and Objects, Destroying Objects
- Accessing attributes, Built-In Class Attributes
- Inheritance and Polymorphism
- Overriding Methods, Data Hiding
- Overloading Operators

Common Errors

Ethan's Learn from experts

➢ Syntax errors, also known as parsing errors, are perhaps the most common kind of complaint you get while you are still learning Python.
➢ Runtime errors, are the errors/exception the program received because of incorrect logic or unexpected scenarios in the programs.
➢ Most runtime exceptions are not handled by programs, so the programs terminate unexpectedly.

For Example:

```
>>> 20 * (10/0)
Traceback (most recent call last):
File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero

>>> 7 + test*3
Traceback (most recent call last): File "<stdin>", line 1, in ?
NameError: name 'spam' is not defined
```

Exceptions

Ethans
Learn from experts

What is exception?

➢ An exception is a Python object that represents an error. Python script encounters a situation that it can't cope with, it raises an exception.

➢ Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them.

1) Exception Handling
2) Assertion

➢ When a Python script raises an exception, it must either handle the exception immediately otherwise it would terminate and come out.

➢There are number of built-in exception Python provides, some of them are listed in next slide.

Built-in Exceptions

Ethans
Learn from experts

```
+-- Exception
    +-- StopIteration
    +-- StandardError
    |    +-- BufferError
    |    +-- ArithmeticError
    |    |    +-- FloatingPointError
    |    |    +-- OverflowError
    |    |    +-- ZeroDivisionError
    |    +-- AssertionError
    |    +-- AttributeError
    |    +-- EnvironmentError
    |    |    +-- IOError
    |    |    +-- OSError
    |    |         +-- WindowsError (Windows)
    |    |         +-- VMSError (VMS)
    |    +-- EOFError
    |    +-- ImportError
    |    +-- LookupError
    |    |    +-- IndexError
    |    |    +-- KeyError
    |    +-- MemoryError
    |    +-- NameError
    |    |    +-- UnboundLocalError
    |    +-- ReferenceError
    |    +-- RuntimeError
    |    |    +-- NotImplementedError
    |    +-- SyntaxError
    |    |    +-- IndentationError
    |    |         +-- TabError
```