

# Programming with Python

File I/O

Exceptions

Modules and Import

# File I/O

# File I/O

- Files are persistent storage

# File I/O

- Files are persistent storage
- Allow data to be stored beyond program lifetime

# File I/O

- Files are persistent storage
- Allow data to be stored beyond program lifetime
- The basic operations on files are
  - open, close, read, write

# File I/O

- Files are persistent storage
- Allow data to be stored beyond program lifetime
- The basic operations on files are
  - open, close, read, write
- Python treat files as sequence of lines
  - sequence operations work for the data read from files

# File I/O: open and close

# File I/O: **open** and **close**

**open**(filename, mode)



# File I/O: **open** and **close**

**open**(filename, mode)

- While opening a file, you need to supply
  - The name of the file, including the path
  - The mode in which you want to open a file
  - Common modes are **r** (read), **w** (write), **a** (append)

# File I/O: **open** and **close**

**open**(filename, mode)

- While opening a file, you need to supply
  - The name of the file, including the path
  - The mode in which you want to open a file
  - Common modes are **r** (read), **w** (write), **a** (append)
- Mode is optional, defaults to **r**

# File I/O: **open** and **close**

**open**(filename, mode)

- While opening a file, you need to supply
  - The name of the file, including the path
  - The mode in which you want to open a file
  - Common modes are **r** (read), **w** (write), **a** (append)
- Mode is optional, defaults to **r**
- **open**(..) returns a file object

# File I/O: **open** and **close**

**open**(filename, mode)

- While opening a file, you need to supply
  - The name of the file, including the path
  - The mode in which you want to open a file
  - Common modes are **r** (read), **w** (write), **a** (append)
- Mode is optional, defaults to **r**
- **open**(..) returns a file object
- **close**() on the file object closes the file
  - finishes any buffered operations

# File I/O: Example

```
>>> players = open('tennis_players', 'w')  
>>>  
>>> • Do some writing  
>>> • How to do it?  
>>>   • see the next few slides  
>>>  
>>> players.close() # done with writing
```

# File I/O: read, write and append

# File I/O: read, write and append

- Reading from an open file returns the contents of the file
  - as **sequence** of lines in the program

# File I/O: **read**, **write** and **append**

- Reading from an open file returns the contents of the file
  - as **sequence** of lines in the program
- Writing to a file
  - **IMPORTANT:** If opened with mode '**w**', clears the existing contents of the file
  - Use append mode ('**a**') to preserve the contents
  - Writing happens at the end



# File I/O: Examples

```
>>> players = open('tennis_players', 'w')
```

```
>>> players.close() # done with writing
```

# File I/O: Examples

```
>>> players = open('tennis_players', 'w')
>>> players.write('Roger Federar\n')
>>> players.write('Rafael Nadal\n')
>>> players.write('Andy Murray\n')
>>> players.write('Novak Djokovic\n')
>>> players.write('Leander Paes\n')
>>> players.close() # done with writing
```

# File I/O: Examples

```
>>> players = open('tennis_players', 'w')
>>> players.write('Roger Federar\n')
>>> players.write('Rafael Nadal\n')
>>> players.write('Andy Murray\n')
>>> players.write('Novak Djokovic\n')
>>> players.write('Leander Paes\n')
>>> players.close() # done with writing

>>> countries = open('tennis_countries', 'w')
>>> countries.write('Switzerland\n')
>>> countries.write('Spain\n')
>>> countries.write('Britain\n')
>>> countries.write('Serbia\n')
>>> countries.write('India\n')

>>> countries.close() # done with writing
```

# File I/O: Examples

```
>>> print(players)
```

# File I/O: Examples

```
>>> print(players)
<closed file 'tennis_players', mode 'w' at 0x
031A48B8>
```

# File I/O: Examples

```
>>> print(players)
<closed file 'tennis_players', mode 'w' at 0x
031A48B8>
>>> print(countries)
<closed file 'tennis_countries', mode 'w' at
0x031A49C0>
```

# File I/O: Examples

```
>>> print(players)
<closed file 'tennis_players', mode 'w' at 0x
031A48B8>
>>> print(countries)
<closed file 'tennis_countries', mode 'w' at
0x031A49C0>

>>> n = open('tennis_players', 'r')
>>> c = open('tennis_countries', 'r')
```

# File I/O: Examples

```
>>> print(players)
<closed file 'tennis_players', mode 'w' at 0x031A48B8>
>>> print(countries)
<closed file 'tennis_countries', mode 'w' at 0x031A49C0>

>>> n = open('tennis_players', 'r')
>>> c = open('tennis_countries', 'r')
>>> n
<open file 'tennis_players', mode 'r' at 0x031A4910>
>>> c
<open file 'tennis_countries', mode 'r' at 0x031A4A70>
```



```
>>> pn = n.read() # read all players
```

```
>>> pn = n.read() # read all players
>>> pn
'Roger Federar\nRafael Nadal\nAndy Murray\nNovak Djokovic\nLeander Paes\n'
```

```
>>> pn = n.read() # read all players
>>> pn
'Roger Federar\nRafael Nadal\nAndy Murray\nNovak Djokovic\nLeander Paes\n'
>>> print(pn)
Roger Federar
Rafael Nadal
Andy Murray
Novak Djokovic
Leander Paes

>>> |
```

```
>>> pn = n.read() # read all players
>>> pn
'Roger Federar\nRafael Nadal\nAndy Murray\nNovak Djokovic\nLeander Paes\n'
>>> print(pn)
Roger Federar
Rafael Nadal
Andy Murray
Novak Djokovic
Leander Paes
>>> |
```



Note empty line due to '\n'

```
>>> pn = n.read() # read all players
>>> pn
'Roger Federar\nRafael Nadal\nAndy Murray\nNovak Djokovic\nLeander Paes\n'
>>> print(pn)
Roger Federar
Rafael Nadal
Andy Murray
Novak Djokovic
Leander Paes
>>> |
>>> n.close()
```



Note empty line due to '\n'

# File I/O: Examples

```
>>> n = open('tennis_players', 'r')  
>>> c = open('tennis_countries', 'r')
```

# File I/O: Examples

```
>>> n = open('tennis_players', 'r')
>>> c = open('tennis_countries', 'r')
>>> pn, pc = [], []
>>> for l in n:
    pn.append(l[:-1]) # ignore '\n'
|>>> n.close()
```

# File I/O: Examples

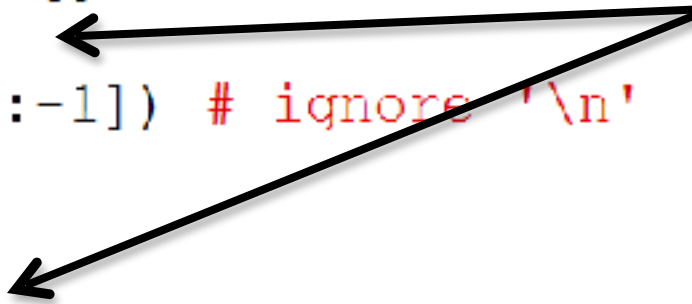
```
>>> n = open('tennis_players', 'r')
>>> c = open('tennis_countries', 'r')
>>> pn, pc = [], []
>>> for l in n:
    pn.append(l[:-1]) # ignore '\n'
>>> n.close()
>>> for l in c:
    pc.append(l[:-1])
>>> c.close()
```



# File I/O: Examples

```
>>> n = open('tennis_players', 'r')
>>> c = open('tennis_countries', 'r')
>>> pn, pc = [], []
>>> for l in n:
    pn.append(l[:-1]) # ignore '\n'
>>> n.close()
>>> for l in c:
    pc.append(l[:-1])
>>> c.close()
```

Note the use of for ... in  
for sequence

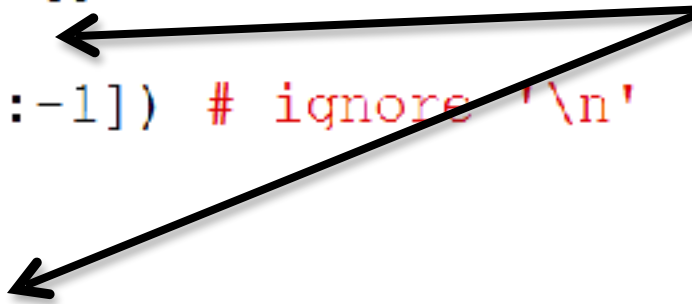


# File I/O: Examples

```
>>> n = open('tennis_players', 'r')
>>> c = open('tennis_countries', 'r')
>>> pn, pc = [], []
>>> for l in n:
    pn.append(l[:-1]) # ignore '\n'
>>> n.close()
>>> for l in c:
    pc.append(l[:-1])
>>> c.close()

>>> print(pn, '\n', pc)
['Roger Federar', 'Rafael Nadal', 'Andy Murra',
'y', 'Novak Djokovic', 'Leander Paes']
['Switzerland', 'Spain', 'Britain', 'Serbia',
'India']
```

Note the use of for ... in  
for sequence



# File I/O: Examples

# File I/O: Examples

```
>>> name_country = []  
>>> for i in range(len(pn)):  
    name_country.append((pn[i], pc[i]))
```

# File I/O: Examples

```
>>> name_country = []
>>> for i in range(len(pn)):
    name_country.append((pn[i], pc[i]))

>>> print(name_country)
[('Roger Federer', 'Switzerland'), ('Rafael N
adal', 'Spain'), ('Andy Murray', 'Britain'),
('Novak Djokovic', 'Serbia'), ('Leander Paes'
, 'India')]
```

# File I/O: Examples

```
>>> name_country = []
>>> for i in range(len(pn)):
    name_country.append((pn[i], pc[i]))

>>> print(name_country)
[('Roger Federar', 'Switzerland'), ('Rafael N
adal', 'Spain'), ('Andy Murray', 'Britain'),
('Novak Djokovic', 'Serbia'), ('Leander Paes'
, 'India')]
>>> n2c = dict(name_country)
>>> print(n2c)
{'Roger Federar': 'Switzerland', 'Andy Murray
': 'Britain', 'Leander Paes': 'India', 'Novak
Djokovic': 'Serbia', 'Rafael Nadal': 'Spain'}
```

# File I/O: Examples

```
>>> name_country = []
>>> for i in range(len(pn)):
    name_country.append((pn[i], pc[i]))


>>> print(name_country)
[('Roger Federar', 'Switzerland'), ('Rafael N
adal', 'Spain'), ('Andy Murray', 'Britain'),
('Novak Djokovic', 'Serbia'), ('Leander Paes'
, 'India')]
>>> n2c = dict(name_country)
>>> print(n2c)
{'Roger Federar': 'Switzerland', 'Andy Murray
': 'Britain', 'Leander Paes': 'India', 'Novak
Djokovic': 'Serbia', 'Rafael Nadal': 'Spain'}
>>> print(n2c['Leander Paes'])
India
```

# Exceptions



# Exceptions

ex|cep|tion

[ɪk'sɛpʃ(ə)n, ɛk'sɛpʃ(ə)n] 

## NOUN

1. a person or thing that is excluded from a general statement or does not follow a rule:

"he always plays top tunes, and tonight was no exception" · [\[more\]](#)

*synonyms:* [anomaly](#) · [irregularity](#) · [deviation](#) · [special case](#) · [\[more\]](#)

Powered by [Oxford Dictionaries](#) · © Oxford University Press · Translation by [Bing Translator](#)

# Exceptions

# Exceptions

- Exceptions are Python's way of telling user that something unexpected has happened

# Exceptions

- Exceptions are Python's way of telling user that something unexpected has happened
- Most often an indication of some failure
  - Access violation (writing to a read-only file)
  - Missing resource (reading a non-existent file)
  - Type incompatibility (multiplying two strings)
  - Bound violation (accessing a string beyond limit)

# Exceptions

- Exceptions are Python's way of telling user that something unexpected has happened
- Most often an indication of some failure
  - Access violation (writing to a read-only file)
  - Missing resource (reading a non-existent file)
  - Type incompatibility (multiplying two strings)
  - Bound violation (accessing a string beyond limit)
- We have seen exceptions in our example

# Exceptions

```
>>> c = open('tennis_countries', 'r')
>>> c.close() # done, close it.
>>> c.read() # Bad read: file already closed
```

# Exceptions

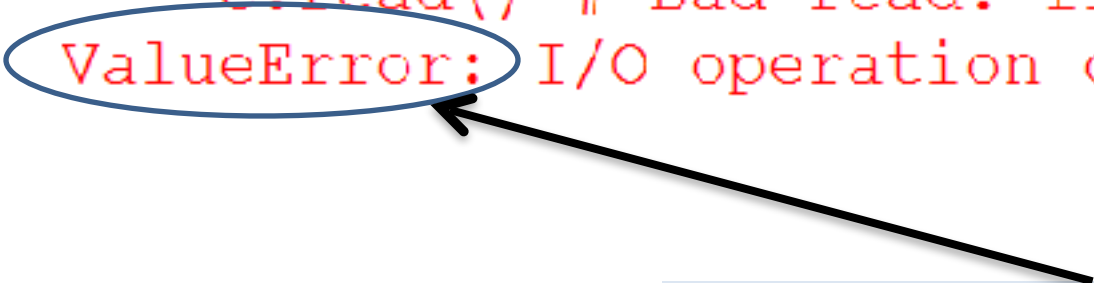
```
>>> c = open('tennis_countries', 'r')
>>> c.close() # done, close it.
>>> c.read() # Bad read: file already closed
```

```
Traceback (most recent call last):
  File "<pyshell#107>", line 1, in <module>
    c.read() # Bad read: file already closed
ValueError: I/O operation on closed file
```

# Exceptions

```
>>> c = open('tennis_countries', 'r')  
>>> c.close() # done, close it.  
>>> c.read() # Bad read: file already closed
```

```
Traceback (most recent call last):  
  File "<pyshell#107>", line 1, in <module>  
    c.read() # Bad read: file already closed  
ValueError: I/O operation on closed file
```



Exception when reading a closed file



# Exceptions can be *Handled*

# Exceptions can be *Handled*

- To recover from unexpected operation

# Exceptions can be *Handled*

- To recover from unexpected operation
- `try ... except ... else ...`

# Exceptions can be *Handled*

- To recover from unexpected operation
- **try ... except ... else ...**

**try:**

Operations that can raise exceptions.

**except** [Optional list of exceptions] :

In case of exceptions, do the recovery.

**else:** # else is optional

If no exception then do normal things.

# Exceptions can be *Handled*

- To recover from unexpected operation
- **try ... except ... else ...**

**try:**

Operations that can raise exceptions.

**except** [Optional list of exceptions] :

In case of exceptions, do the recovery.

**else:** # else is optional

If no exception then do normal things.

- **try ... finally ...**

# Exceptions can be *Handled*

- To recover from unexpected operation
- **try ... except ... else ...**

```
try:
    Operations that can raise exceptions.
except [Optional list of exceptions] :
    In case of exceptions, do the recovery.
else: # else is optional
    If no exception then do normal things.
```

- **try ... finally ...**

```
try:
    Operations that can raise exceptions.
finally:
    Execute irrespective of whether exception
    was raised or not. Typically clean-up stuff.
```

# Example

```
try:
    w = open('/target.txt', 'w')
except: # IO Error
    print('Can not write to the target file' )
else: # Note: else is with except, not with if
    w.write('success')
    print('Can write in ROOT directory!' )
    w.close()
```

# More about try...except



# More about try...except

- A **try** statement may have more than one **except** clause
  - to specify handlers for different exceptions.

# More about try...except

- A **try** statement may have more than one **except** clause
  - to specify handlers for different exceptions.
- At most one handler will be executed.

# More about try...except

- A **try** statement may have more than one **except** clause
  - to specify handlers for different exceptions.
- At most one handler will be executed.
- An **except** clause may name multiple exceptions as a parenthesized tuple.

# More about try...except

- A **try** statement may have more than one **except** clause
  - to specify handlers for different exceptions.
- At most one handler will be executed.
- An **except** clause may name multiple exceptions as a parenthesized tuple.
- The last **except** clause may omit the exception name
  - Catches *any* exception

# Modules

# Modules

- As program gets longer, need to organize them for easier access and easier maintenance.

# Modules

- As program gets longer, need to organize them for easier access and easier maintenance.
- Reuse same functions across programs without copying its definition into each program.

# Modules

- As program gets longer, need to organize them for easier access and easier maintenance.
- Reuse same functions across programs without copying its definition into each program.
- Python allows putting definitions in a file
  - use them in a script or in an interactive instance of the interpreter



# Modules

- As program gets longer, need to organize them for easier access and easier maintenance.
- Reuse same functions across programs without copying its definition into each program.
- Python allows putting definitions in a file
  - use them in a script or in an interactive instance of the interpreter
- Such a file is called a *module*
  - definitions from a module can be *imported* into other modules or into the *main* module

# Modules

# Modules

- A module is a file containing Python definitions and statements.

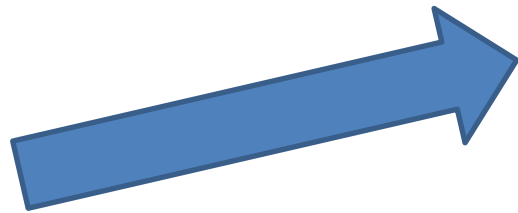
# Modules

- A module is a file containing Python definitions and statements.
- The file name is the module name with the suffix `.py` appended.

# Modules

- 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 is available in the global variable `__name__`.

# Modules Example



fib.py - C:\

fib.py - C:\Users\karkare\Google Drive\IITK\Courses\2016Python\Programs\fib.py (2.7.12)

File Edit Format Run Options Window Help

```
# Module for fibonacci numbers
```

```
def fib_rec(n):  
    '''recursive fibonacci'''  
    if (n <= 1):  
        return n  
    else:  
        return fib_rec(n-1) + fib_rec(n-2)
```

# Modules Example

```
def fib_rec(n):  
    '''recursive fibonacci'''  
    if (n <= 1):  
        return n  
    else:  
        return fib_rec(n-1) + fib_rec(n-2)  
  
def fib_iter(n):  
    '''iterative fibonacci'''  
    cur, nxt = 0, 1  
    for k in range(n):  
        cur, nxt = nxt, cur+nxt  
    return cur  
  
def fib_upto(n):  
    '''given n, return list of fibonacci  
    numbers < n'''  
    cur, nxt = 0, 1  
    lst = []  
    while (cur < n):  
        lst.append(cur)  
        cur, nxt = nxt, cur+nxt  
    return lst
```

# Modules Example

```
def fib_rec(n):  
    '''recursive fibonacci'''  
    if (n <= 1):  
        return n  
    else:  
        return fib_rec(n-1) + fib_rec(n-2)  
  
def fib_iter(n):  
    '''iterative fibonacci'''  
    cur, nxt = 0, 1  
    for k in range(n):  
        cur, nxt = nxt, cur+nxt  
    return cur  
  
def fib_upto(n):  
    '''given n, return list of fibonacci  
    numbers < n'''  
    cur, nxt = 0, 1  
    lst = []  
    while (cur < n):  
        lst.append(cur)  
        cur, nxt = nxt, cur+nxt  
    return lst
```

```
>>> import fib  
>>> fib.fib_upto(5)  
[0, 1, 1, 2, 3]
```



# Modules Example

```
def fib_rec(n):  
    '''recursive fibonacci'''  
    if (n <= 1):  
        return n  
    else:  
        return fib_rec(n-1) + fib_rec(n-2)  
  
def fib_iter(n):  
    '''iterative fibonacci'''  
    cur, nxt = 0, 1  
    for k in range(n):  
        cur, nxt = nxt, cur+nxt  
    return cur  
  
def fib_upto(n):  
    '''given n, return list of fibonacci  
    numbers < n'''  
    cur, nxt = 0, 1  
    lst = []  
    while (cur < n):  
        lst.append(cur)  
        cur, nxt = nxt, cur+nxt  
    return lst
```

```
>>> import fib  
>>> fib.fib_upto(5)  
[0, 1, 1, 2, 3]  
  
>>> fib.fib_rec(10)  
55  
  
>>> fib.fib_iter(20)  
6765
```

# Modules Example


```
def fib_rec(n):  
    '''recursive fibonacci'''  
    if (n <= 1):  
        return n  
    else:  
        return fib_rec(n-1) + fib_rec(n-2)  
  
def fib_iter(n):  
    '''iterative fibonacci'''  
    cur, nxt = 0, 1  
    for k in range(n):  
        cur, nxt = nxt, cur+nxt  
    return cur  
  
def fib_upto(n):  
    '''given n, return list of fibonacci  
    numbers < n'''  
    cur, nxt = 0, 1  
    lst = []  
    while (cur < n):  
        lst.append(cur)  
        cur, nxt = nxt, cur+nxt  
    return lst
```

```
>>> import fib  
>>> fib.fib_upto(5)  
[0, 1, 1, 2, 3]  
  
>>> fib.fib_rec(10)  
55  
  
>>> fib.fib_iter(20)  
6765  
  
>>> fib.__name__  
'fib'
```

# Modules Example

```
def fib_rec(n):  
    '''recursive fibonacci'''  
    if (n <= 1):  
        return n  
    else:  
        return fib_rec(n-1) + fib_rec(n-2)  
  
def fib_iter(n):  
    '''iterative fibonacci'''  
    cur, nxt = 0, 1  
    for k in range(n):  
        cur, nxt = nxt, cur+nxt  
    return cur  
  
def fib_upto(n):  
    '''given n, return list of fibonacci  
    numbers < n'''  
    cur, nxt = 0, 1  
    lst = []  
    while (cur < n):  
        lst.append(cur)  
        cur, nxt = nxt, cur+nxt  
    return lst
```

```
>>> import fib  
>>> fib.fib_upto(5)  
[0, 1, 1, 2, 3]  
  
>>> fib.fib_rec(10)  
55  
  
>>> fib.fib_iter(20)  
6765  
  
>>> fib.__name__  
'fib'
```



Within a module, the module's name is available as the value of the global variable **\_\_name\_\_**.

# Importing Specific Functions

# Importing Specific Functions

- To import specific functions from a module

# Importing Specific Functions

- To import specific functions from a module

```
>>> from fib import fib_upto
```

# Importing Specific Functions

- To import specific functions from a module

```
>>> from fib import fib_upto  
>>> fib_upto(6)  
[0, 1, 1, 2, 3, 5]
```

# Importing Specific Functions

- To import specific functions from a module

```
>>> from fib import fib_upto
>>> fib_upto(6)
[0, 1, 1, 2, 3, 5]
>>> fib_iter(1)
```



# Importing Specific Functions

- To import specific functions from a module

```
>>> from fib import fib_upto
>>> fib_upto(6)
[0, 1, 1, 2, 3, 5]
>>> fib_iter(1)
```

```
Traceback (most recent call last):
```

```
  File "<pyshell#16>", line 1, in <module>
    fib_iter(1)
```

```
NameError: name 'fib_iter' is not defined
```

# Importing Specific Functions

- To import specific functions from a module

```
>>> from fib import fib_upto
>>> fib_upto(6)
[0, 1, 1, 2, 3, 5]
>>> fib_iter(1)
```

```
Traceback (most recent call last):
```

```
File "<pyshell#16>", line 1, in <module>
    fib_iter(1)
```

```
NameError: name 'fib_iter' is not defined
```

- This brings only the imported functions in the current symbol table
  - No need of **modulename**. (absence of `fib.` in the example)

# Importing ALL Functions

```
>>> from fib import *  
>>> fib_upto(6)  
[0, 1, 1, 2, 3, 5]  
>>> fib_iter(8)  
21
```

# Importing ALL Functions

- To import *all* functions from a module, in the current symbol table

```
>>> from fib import *
>>> fib_upto(6)
[0, 1, 1, 2, 3, 5]
>>> fib_iter(8)
21
```

# Importing ALL Functions

- To import *all* functions from a module, in the current symbol table

```
>>> from fib import *  
>>> fib_upto(6)  
[0, 1, 1, 2, 3, 5]  
>>> fib_iter(8)  
21  
.
```

- This imports all names **except those beginning with an underscore (\_).**

# `__main__` in Modules

# \_\_main\_\_ in Modules

- When you run a module on the command line with  
`python fib.py <arguments>`

# \_\_main\_\_ in Modules

- When you run a module on the command line with  
`python fib.py <arguments>`  
the code in the module will be executed, just as if



# \_\_main\_\_ in Modules

- When you run a module on the command line with `python fib.py <arguments>` the code in the module will be executed, just as if you imported it, but with the `__name__` set to

# \_\_main\_\_ in Modules

- When you run a module on the command line with `python fib.py <arguments>` the code in the module will be executed, just as if you imported it, but with the `__name__` set to `"__main__"`.

# \_\_main\_\_ in Modules

- When you run a module on the command line with `python fib.py <arguments>` the code in the module will be executed, just as if you imported it, but with the `__name__` set to `"__main__"`.

- By adding this code at the end of your module

```
if __name__ == "__main__":  
    ... # Some code here
```

# \_\_main\_\_ in Modules

- When you run a module on the command line with `python fib.py <arguments>` the code in the module will be executed, just as if you imported it, but with the `__name__` set to `"__main__"`.
- By adding this code at the end of your module  

```
if __name__ == "__main__":  
    ... # Some code here
```

you can make the file usable as a script as well as an

# \_\_main\_\_ in Modules

- When you run a module on the command line with `python fib.py <arguments>` the code in the module will be executed, just as if you imported it, but with the `__name__` set to `"__main__"`.
- By adding this code at the end of your module  

```
if __name__ == "__main__":  
    ... # Some code here
```

you can make the file usable as a script as well as an importable module

# `__main__` in Modules

# \_\_main\_\_ in Modules

```
if __name__ == "__main__":  
    import sys  
    print (fib_iter(int(sys.argv[1])))
```

# \_\_main\_\_ in Modules

```
if __name__ == "__main__":  
    import sys  
    print (fib_iter(int(sys.argv[1])))
```

- This code parses the command line only if the module is executed as the “main” file:

```
$ python fib.py 10  
55
```



# \_\_main\_\_ in Modules

```
if __name__ == "__main__":  
    import sys  
    print (fib_iter(int(sys.argv[1])))
```

- This code parses the command line only if the module is executed as the “main” file:

```
$ python fib.py 10  
55
```

- If the module is imported, the code is not run:

# \_\_main\_\_ in Modules

```
if __name__ == "__main__":  
    import sys  
    print (fib_iter(int(sys.argv[1])))
```

- This code parses the command line only if the module is executed as the “main” file:

```
$ python fib.py 10  
55
```

- If the module is imported, the code is not run:

```
>>> import fib
```