File I/O

File I/O

and understanding primary/secondary memory

Review of Python Memory

Review of Python Memory

Actual footage of a real-life variable being declared (colorized):

Review of Python Memory

Actual footage of a real-life variable being declared (colorized):

```
>>> x = 1
```

Review of Python Memory

Actual footage of a real-life variable being declared (colorized):

```
>>> x = 1
>>>
```

Review of Python Memory

Actual footage of a real-life variable being declared (colorized):

```
>>> x = 1
>>>
```

Memory:

Address	Contents
00000000	01101011
00000001	11001100
00000010	10001000
00000011	10101110
⋮	⋮
11111111	00100000

Review of Python Memory

Actual footage of a real-life variable being declared (colorized):

```
>>> x = 1
>>>
```

$\rightarrow$ `Memory:`

Address	Contents
00000000	01101011
00000001	11001100
00000010	10001000
00000011	10101110
⋮	⋮
11111111	00100000

Review of Python Memory

Actual footage of a real-life variable being declared (colorized):

```
>>> x = 1
>>>
```

→

`Memory:`

Address	Contents
00000000	01101011
00000001	11001100
00000010	10001000
00000011	10101110
⋮	⋮
11111111	00100000

Review of Python Memory

Actual footage of a real-life variable being declared (colorized):

```
>>> x = 1
>>>
```

→

`Memory:`

Address	Contents
00000000	01101011
00000001	11001100
x	**00000001**
00000011	10101110
⋮	⋮
11111111	00100000

Review of Python Memory (continued)

Actual footage of a real-life variable being updated (colorized):

```
>>> x = 1
>>>
```

Memory:

Address	Contents
00000000	01101011
00000001	11001100
x	**00000001**
00000011	10101110
⋮	⋮
11111111	00100000

Review of Python Memory (continued)

Actual footage of a real-life variable being updated (colorized):

```
>>> x = 1
>>> x = 2
```

Memory:

Address	Contents
00000000	01101011
00000001	11001100
x	**00000001**
00000011	10101110
⋮	⋮
11111111	00100000

Review of Python Memory (continued)

Actual footage of a real-life variable being updated (colorized):

```
>>> x = 1
>>> x = 2
```

haha, you're a 2 now

Cpt. Python

`Memory:`

Address	Contents
00000000	01101011
00000001	11001100
x	**00000001**
00000011	10101110
⋮	⋮
11111111	00100000

Review of Python Memory (continued)

Actual footage of a real-life variable being updated (colorized):

```
>>> x = 1
>>> x = 2
```

haha, you're a 2 now

Cpt. Python

`Memory:`

Address	Contents
00000000	01101011
00000001	11001100
x	**00000010**
00000011	10101110
⋮	⋮
11111111	00100000

Review of Python Memory (continued)

Actual footage of a real-life variable being updated (again):

```
>>> x = 1
>>> x = 2
>>>
```

Memory:

Address	Contents
00000000	01101011
00000001	11001100
x	**00000010**
00000011	10101110
⋮	⋮
11111111	00100000

Review of Python Memory (continued)

Actual footage of a real-life variable being updated (again):

```
>>> x = 1
>>> x = 2
>>> x = x + 1
```

Memory:

Address	Contents
00000000	01101011
00000001	11001100
x	**00000010**
00000011	10101110
⋮	⋮
11111111	00100000

Review of Python Memory (continued)

Actual footage of a real-life variable being updated (again):

```
>>> x = 1
>>> x = 2
>>> x = x + 1
```

Memory:

Address	Contents
00000000	01101011
00000001	11001100
x	**00000010**
00000011	10101110
⋮	⋮
11111111	00100000

Review of Python Memory (continued)

Actual footage of a real-life variable being updated (again):

```
>>> x = 1
>>> x = 2
>>> x = x + 1
```

Memory:

Address	Contents
00000000	01101011
00000001	11001100
x	**00000011**
00000011	10101110
⋮	⋮
11111111	00100000

Review of Python Memory (continued)

Actual footage of a real-life variable being released:

```
>>> x = 1
>>> x = 2
>>> x = x + 1
>>>
```

Memory:

Address	Contents
00000000	01101011
00000001	11001100
x	**00000011**
00000011	10101110
⋮	⋮
11111111	00100000

Review of Python Memory (continued)

Actual footage of a real-life variable being released:

```
>>> x = 1
>>> x = 2
>>> x = x + 1
>>> quit()
```

Memory:

Address	Contents
00000000	01101011
00000001	11001100
x	**00000011**
00000011	10101110
⋮	⋮
11111111	00100000

Review of Python Memory (continued)

Actual footage of a real-life variable being released:

```
>>> x = 1
>>> x = 2
>>> x = x + 1
>>> quit()
```

Memory:

Address	Contents
00000000	01101011
00000001	11001100
x	**00000011**
00000011	10101110
⋮	⋮
11111111	00100000

Review of Python Memory (continued)

Actual footage of a real-life variable being released:

```
>>> x = 1
>>> x = 2
>>> x = x + 1
>>> quit()
```

`Memory:`

Address	Contents
00000000	01101011
00000001	11001100
00000010	00000011
00000011	10101110
⋮	⋮
11111111	00100000

Volatile Memory

Volatile Memory

Python stores variables in your computer's *primary memory*

Volatile Memory

Python stores variables in your computer's *primary memory*

Primary memory is **fast** — it is maintained by electrical currents

Volatile Memory

Python stores variables in your computer's *primary memory*

Primary memory is **fast** — it is maintained by electrical currents

Primary memory is usually *volatile* — when you exit the program or cut the power, it is lost forever

Volatile Memory

Python stores variables in your computer's *primary memory*

Primary memory is **fast** — it is maintained by electrical currents

Primary memory is usually *volatile* — when you exit the program or cut the power, it is lost forever

In your desktop computer or laptop, primary memory = RAM

Non-volatile Memory

Non-volatile Memory

To store data for the long-term, you need to save a *file* in your *secondary memory*

Non-volatile Memory

To store data for the long-term, you need to save a *file* in your *secondary memory*

Secondary memory is **slow** — requires physically moving or realigning components

Non-volatile Memory

To store data for the long-term, you need to save a *file* in your *secondary memory*

Secondary memory is **slow** — requires physically moving or realigning components

Secondary memory is usually *non-volatile* — it stays, even after you cut the power

Non-volatile Memory

To store data for the long-term, you need to save a *file* in your *secondary memory*

Secondary memory is **slow** — requires physically moving or realigning components

Secondary memory is usually *non-volatile* — it stays, even after you cut the power

In a computer or laptop, secondary memory = disk, thumb drive, or CD

Primary vs. Secondary Memory

Primary vs. Secondary Memory

Primary memory

Secondary memory

Primary vs. Secondary Memory

Primary memory

RAM
global variables local variables named constants arguments return values temp values

Secondary memory

Primary vs. Secondary Memory

Primary memory

RAM
global variables
local variables
named constants
arguments
return values
temp values

Secondary memory

Disk
text files
pdf files
images
audio files
python scripts
save files

Concept Check!

Which of the following are saved in secondary *non-volatile* memory?

1. A plain text file?
2. A Python variable: `x = 3`?
3. A Python script?
4. A digital photograph?
5. A Python dictionary:
   `myDict = {"item": "beans", "price": 1.99}`?
6. An mp3 file?

Concept Check!

Which of the following are saved in secondary *non-volatile* memory?

1. A plain text file? ✓
2. A Python variable: `x = 3`?
3. A Python script?
4. A digital photograph?
5. A Python dictionary:
   `myDict = {"item": "beans", "price": 1.99}`?
6. An mp3 file?

Concept Check!

Which of the following are saved in secondary *non-volatile* memory?

1. A plain text file? ✓
2. A Python variable: `x = 3`? ✗
3. A Python script?
4. A digital photograph?
5. A Python dictionary:
   `myDict = {"item": "beans", "price": 1.99}`?
6. An mp3 file?

Concept Check!

Which of the following are saved in secondary *non-volatile* memory?

1. A plain text file? ✓
2. A Python variable: `x = 3`? ✗
3. A Python script? ✓
4. A digital photograph?
5. A Python dictionary:
   `myDict = {"item": "beans", "price": 1.99}`?
6. An mp3 file?

Concept Check!

Which of the following are saved in secondary *non-volatile* memory?

1. A plain text file? ✓
2. A Python variable: `x = 3`? ✗
3. A Python script? ✓
4. A digital photograph? ✓
5. A Python dictionary:
   `myDict = {"item": "beans", "price": 1.99}`?
6. An mp3 file?

Concept Check!

Which of the following are saved in secondary *non-volatile* memory?

1. A plain text file? ✓
2. A Python variable: `x = 3`? ✗
3. A Python script? ✓
4. A digital photograph? ✓
5. A Python dictionary: ✗
   `myDict = {"item": "beans", "price": 1.99}`?
6. An mp3 file?

Concept Check!

Which of the following are saved in secondary *non-volatile* memory?

1. A plain text file? ✓
2. A Python variable: `x = 3`? ✗
3. A Python script? ✓
4. A digital photograph? ✓
5. A Python dictionary: ✗
   `myDict = {"item": "beans", "price": 1.99}`?
6. An mp3 file? ✓

How to save a file in Python?

How to save a file in Python?

- `myFile = open("filename.txt", "w")` – “w” = write a new file

How to save a file in Python?

- `myFile = open("filename.txt", "w")` – “w” = write a new file
- `myFile.write("Any string you'd like to write")`

How to save a file in Python?

- `myFile = open("filename.txt", "w")` – “w” = write a new file
- `myFile.write("Any string you'd like to write")`
- `myFile.write(str(myVar))`

How to save a file in Python?

- `myFile = open("filename.txt", "w")` – “w” = write a new file
- `myFile.write("Any string you'd like to write")`
- `myFile.write(str(myVar))`
- `myFile.close()`

How to save a file in Python?

- `myFile = open("filename.txt", "w")` – "w" = write a new file
- `myFile.write("Any string you'd like to write")`
- `myFile.write(str(myVar))`
- `myFile.close()`
- `myFile = open("filename.txt", "a")` – "a" = append to existing file

How to read a file in Python?

How to read a file in Python?

- `myFile = open("filename.txt", "r")` – “r” = read a file

How to read a file in Python?

- `myFile = open("filename.txt", "r")` – "r" = read a file
- `myVar1 = myFile.readline()`

How to read a file in Python?

- `myFile = open("filename.txt", "r")` – “r” = read a file
- `myVar1 = myFile.readline()`
- `myVar2 = myFile.readline()`

How to read a file in Python?

- `myFile = open("filename.txt", "r")` – “r” = read a file
- `myVar1 = myFile.readline()`
- `myVar2 = myFile.readline()`
- `myFile.close()`

Read in a loop

Read in a loop

- `myList = []`

Read in a loop

- `myList = []`
- `myFile = open("filename.txt", "r")`

Read in a loop

- `myList = []`
- `myFile = open("filename.txt", "r")`
-
```
for line in myFile:
    myList.append(line)
```

Read in a loop

- `myList = []`
- `myFile = open("filename.txt", "r")`
-
```
for line in myFile:
    myList.append(line)
```
- `myFile.close()`

Safely reading

Safely reading

If an error occurs while reading, then your files won't be closed

Safely reading

If an error occurs while reading, then your files won't be closed

Use a `with` clause to have them automatically close:

Safely reading

If an error occurs while reading, then your files won't be closed

Use a `with` clause to have them automatically close:

```
myList = []
with open("filename.txt", "r") as myFile:
    for line in myFile:
        myList.append(line)
```

More about files

https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files