STAT606
Computing for Data Science and Statistics

Lecture 6: Python on the Command Line

Scripting with Python

Python is an ideal **scripting language**

- Simple to read/write code
- Good support for processing strings and files

Scripting with Python

Python is an ideal **scripting language**

- Simple to read/write code
- Good support for processing strings and files

We can open the Python interpreter from the command line:

```
keith@Steinhaus:~/demo$ python3
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 03:03:55)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> print('This is the Python interpreter')
This is the Python interpreter
>>>
```

Scripting with Python

We can also write Python code, and then run it from the command line.

```
keith@Steinhaus:~/demo$ ls
print_me.py
keith@Steinhaus:~/demo$ cat print_me.py
print('This is a Python script!')
print('Run it on the command line!')
keith@Steinhaus:~/demo$ python3 print_me.py
This is a Python script!
Run it on the command line!
keith@Steinhaus:~$
```

print_me.py

```
print('This is a Python script!')
print('Run it on the command line!')
```

Scripting with Python

We can also write Python code, and then run it from the command line.

```
keith@Steinhaus:~/demo$ ls
print_me.py
keith@Steinhaus:~/demo$ cat print_me.py
print('This is a Python script!')
print('Run it on the command line!')
keith@Steinhaus:~/demo$ python3 print_me.py
This is a Python script!
Run it on the command line!
keith@Steinhaus:~$
```

`python3 file.py` tells Python to run the Python code in `file.py`, line by line. Anything Python prints is sent to `stdout`.

print_me.py

```
print('This is a Python script!')
print('Run it on the command line!')
```

Accessing Command-Line Arguments: `sys`

`sys` module gives us access to certain system-level variables

e.g., information about environment we are running in

https://docs.python.org/3/library/sys.html

Most important for us: `sys.argv`

https://docs.python.org/3/library/sys.html#sys.argv

`sys.argv` is a list, whose elements are the command-line arguments

`sys.argv[0]` is script name, `sys.argv[1]` is first argument, etc.

Should look familiar from Bash scripting!

Accessing Command-Line Arguments: `sys`

`sys` module gives us access to certain system-level variables

e.g., information about environment we are running in

https://docs.python.org/3/library/sys.html

Most important for us: `sys.argv`

```
keith@Steinhaus:~/demo$ python print_args.py cat dog bird goat
print_args.py
cat
dog
bird
goat
keith@Steinhaus:~/demo$
```

print_args.py

```
import sys

for a in sys.argv:
    print(a)
```

Accessing Command-Line Arguments: `sys`

`sys` module gives us access to certain system-level variables

e.g., information about environment we are running in

https://docs.python.org/3/library/sys.html

Most important for us: `sys.argv`

`sys.argv` is a list, whose elements are command line arguments.

```
keith@Steinhaus:~/demo$ python print_args.py cat dog bird goat
print_args.py
cat
dog
bird
goat
keith@Steinhaus:~/demo$
```

print_args.py

```
import sys

for a in sys.argv:
    print(a)
```

Accessing Command-Line Arguments: `sys`

`sys` module gives us access to certain system-level variables

e.g., information about environment we are running in

https://docs.python.org/3/library/sys.html

Most important for us: `sys.argv`

```
keith@Steinhaus:~/demo$ python print_args.py cat dog bird goat
print_args.py
cat
dog
bird
goat
keith@Steinhaus:~/demo$
```

Script name is the first argument passed to Python, so `sys.argv`[0] is the script name.

print_args.py

```
import sys

for a in sys.argv:
    print(a)
```

Checking `__main__`

We can write modules for import

Or we can write for command line

But can we do both?

prime.py

```
1  import math
2
3  def is_prime(n):
4      if n <= 1:
5          return False
6      elif n==2:
7          return True
8      else:
9          ulim = math.ceil(math.sqrt(n))
10         for k in range(2,ulim+1):
11             if n%k==0:
12                 return False
13         return True
14
15 if __name__ == "__main__":
16     import sys
17     USAGE='python ' + sys.argv[0] + ' int'
18     if len(sys.argv) != 2:
19         raise RuntimeError(USAGE)
20     else:
21         n = int( sys.argv[1] )
22         if is_prime(n):
23             print(sys.argv[1]+' is prime.')
24         else:
25             print(sys.argv[1]+' is not prime.')
```

Checking `__main__`

We can write modules for import

Or we can write for command line

But can we do both?

Yes! By checking the namespace

prime.py

```
1  import math
2
3  def is_prime(n):
4      if n <= 1:
5          return False
6      elif n==2:
7          return True
8      else:
9          ulim = math.ceil(math.sqrt(n))
10         for k in range(2,ulim+1):
11             if n%k==0:
12                 return False
13         return True
14
15 if __name__ == "__main__":
16     import sys
17     USAGE='python ' + sys.argv[0] + ' int'
18     if len(sys.argv) != 2:
19         raise RuntimeError(USAGE)
20     else:
21         n = int( sys.argv[1] )
22         if is_prime(n):
23             print(sys.argv[1]+' is prime.')
24         else:
25             print(sys.argv[1]+' is not prime.')
```

Checking `__main__`

This block of code only runs if we are in the `__main__` namespace.

prime.py

```python
import math

def is_prime(n):
    if n <= 1:
        return False
    elif n==2:
        return True
    else:
        ulim = math.ceil(math.sqrt(n))
        for k in range(2,ulim+1):
            if n%k==0:
                return False
        return True

if __name__ == "__main__":
    import sys
    USAGE='python ' + sys.argv[0] + ' int'
    if len(sys.argv) != 2:
        raise RuntimeError(USAGE)
    else:
        n = int( sys.argv[1] )
        if is_prime(n):
            print(sys.argv[1]+' is prime.')
        else:
            print(sys.argv[1]+' is not prime.')
```

Checking `__main__`

This block of code only runs if we are in the `__main__` namespace.

```
import prime

prime.is_prime(8675309)
```

```
True
```

When we import a module, the code in that module runs in its own local scope, so `__name__ != "__main__"`

prime.py

```
import math

def is_prime(n):
    if n <= 1:
        return False
    elif n==2:
        return True
    else:
        ulim = math.ceil(math.sqrt(n))
        for k in range(2,ulim+1):
            if n%k==0:
                return False
        return True

if __name__ == "__main__":
    import sys
    USAGE='python ' + sys.argv[0] + ' int'
    if len(sys.argv) != 2:
        raise RuntimeError(USAGE)
    else:
        n = int( sys.argv[1] )
        if is_prime(n):
            print(sys.argv[1]+' is prime.')
        else:
            print(sys.argv[1]+' is not prime.')
```

Checking `__main__`

This block of code only runs if we are in the `__main__` namespace.

```
1  import prime
2
3  prime.is_prime(8675309)
```

```
True
```

“The import statement combines two operations; it searches for the named module, then it binds the results of that search to a name in the local scope.” https://docs.python.org/3/reference/import.html

prime.py

```
1  import math
2
3  def is_prime(n):
4      if n <= 1:
5          return False
6      elif n==2:
7          return True
8      else:
9          ulim = math.ceil(math.sqrt(n))
10         for k in range(2,ulim+1):
11             if n%k==0:
12                 return False
13         return True
14
15 if __name__ == "__main__":
16     import sys
17     USAGE='python ' + sys.argv[0] + ' int'
18     if len(sys.argv) != 2:
19         raise RuntimeError(USAGE)
20     else:
21         n = int( sys.argv[1] )
22         if is_prime(n):
23             print(sys.argv[1]+' is prime.')
24         else:
25             print(sys.argv[1]+' is not prime.')
```

Checking `__main__`

```
k@S:~$ ls
prime.py
k@S:~$ python prime.py 8675309
8675309 is prime.
k@S:~/$
k@S:~/$ python prime.py 10 20
Traceback (most recent call
last):
  File "prime.py", line 19, in
<module>
    raise RuntimeError(USAGE)
RuntimeError: python prime.py int
k@S:~/$
```

prime.py

```
1  import math
2
3  def is_prime(n):
4      if n <= 1:
5          return False
6      elif n==2:
7          return True
8      else:
9          ulim = math.ceil(math.sqrt(n))
10         for k in range(2,ulim+1):
11             if n%k==0:
12                 return False
13         return True
14
15 if __name__ == "__main__":
16     import sys
17     USAGE='python ' + sys.argv[0] + ' int'
18     if len(sys.argv) != 2:
19         raise RuntimeError(USAGE)
20     else:
21         n = int( sys.argv[1] )
22         if is_prime(n):
23             print(sys.argv[1]+' is prime.')
24         else:
25             print(sys.argv[1]+' is not prime.')
```

Checking `__main__`

```
k@S:~$ ls
prime.py
k@S:~$ python prime.py 8675309
8675309 is prime.
k@S:~/$
k@S:~/$ python prime.py 10 20
Traceback (most recent call
last):
  File "prime.py", line 19, in
<module>
    raise RuntimeError(USAGE)
RuntimeError: python prime.py int
k@S:~/$
```

prime.py

```
1  import math
2
3  def is_prime(n):
4      if n <= 1:
5          return False
6      elif n==2:
7          return True
8      else:
9          ulim = math.ceil(math.sqrt(n))
10         for k in range(2,ulim+1):
11             if n%k==0:
12                 return False
13         return True
14
15 if __name__ == "__main__":
16     import sys
17     USAGE='python ' + sys.argv[0] + ' int'
18     if len(sys.argv) != 2:
19         raise RuntimeError(USAGE)
20     else:
21         n = int( sys.argv[1] )
22         if is_prime(n):
23             print(sys.argv[1]+' is prime.')
24         else:
25             print(sys.argv[1]+' is not prime.')
```

Define `is_prime()`.

Checking `__main__`

```
k@S:~$ ls
prime.py
k@S:~$ python prime.py 8675309
8675309 is prime.
k@S:~/$
k@S:~/$ python prime.py 10 20
Traceback (most recent call
last):
  File "prime.py", line 19, in
<module>
    raise RuntimeError(USAGE)
RuntimeError: python prime.py int
k@S:~/$
```

On the command line, we are in the main namespace.

prime.py

```
1  import math
2
3  def is_prime(n):
4      if n <= 1:
5          return False
6      elif n==2:
7          return True
8      else:
9          ulim = math.ceil(math.sqrt(n))
10         for k in range(2,ulim+1):
11             if n%k==0:
12                 return False
13         return True
14
15 if __name__ == "__main__":
16     import sys
17     USAGE='python ' + sys.argv[0] + ' int'
18     if len(sys.argv) != 2:
19         raise RuntimeError(USAGE)
20     else:
21         n = int( sys.argv[1] )
22         if is_prime(n):
23             print(sys.argv[1]+' is prime.')
24         else:
25             print(sys.argv[1]+' is not prime.')
```

Checking `__main__`

```
k@S:~$ ls
prime.py
k@S:~$ python prime.py 8675309
8675309 is prime.
k@S:~/$
k@S:~/$ python prime.py 10 20
Traceback (most recent call
last):
  File "prime.py", line 19, in
<module>
    raise RuntimeError(USAGE)
RuntimeError: python prime.py int
k@S:~/$
```

Import `sys` module so we can access command line arguments.

prime.py

```
1  import math
2
3  def is_prime(n):
4      if n <= 1:
5          return False
6      elif n==2:
7          return True
8      else:
9          ulim = math.ceil(math.sqrt(n))
10         for k in range(2,ulim+1):
11             if n%k==0:
12                 return False
13         return True
14
15 if __name__ == "__main__":
16     import sys
17     USAGE= 'python ' + sys.argv[0] + ' int'
18     if len(sys.argv) != 2:
19         raise RuntimeError(USAGE)
20     else:
21         n = int( sys.argv[1] )
22         if is_prime(n):
23             print(sys.argv[1]+' is prime.')
24         else:
25             print(sys.argv[1]+' is not prime.')
```

Checking `__main__`

```
k@S:~$ ls
prime.py
k@S:~$ python prime.py 8675309
8675309 is prime.
k@S:~/$
k@S:~/$ python prime.py 10 20
Traceback (most recent call
last):
  File "prime.py", line 19, in
<module>
    raise RuntimeError(USAGE)
RuntimeError: python prime.py int
k@S:~/$
```

If we don't get exactly one command line argument, raise an error.

prime.py

```
1  import math
2
3  def is_prime(n):
4      if n <= 1:
5          return False
6      elif n==2:
7          return True
8      else:
9          ulim = math.ceil(math.sqrt(n))
10         for k in range(2,ulim+1):
11             if n%k==0:
12                 return False
13         return True
14
15 if __name__ == "__main__":
16     import sys
17     USAGE='python ' + sys.argv[0] + ' int'
18     if len(sys.argv) != 2:
19         raise RuntimeError(USAGE)
20     else:
21         n = int( sys.argv[1] )
22         if is_prime(n):
23             print(sys.argv[1]+' is prime.')
24         else:
25             print(sys.argv[1]+' is not prime.')
```

Checking `__main__`

```
k@S:~$ ls
prime.py
k@S:~$ python prime.py 8675309
8675309 is prime.
k@S:~/$
k@S:~/$ python prime.py 10 20
Traceback (most recent call
last):
  File "prime.py", line 19, in
<module>
    raise RuntimeError(USAGE)
RuntimeError: python prime.py int
k@S:~/$
```

Arguments come in as strings. Cast to an integer and pass to `is_prime()`.

prime.py

```
1  import math
2
3  def is_prime(n):
4      if n <= 1:
5          return False
6      elif n==2:
7          return True
8      else:
9          ulim = math.ceil(math.sqrt(n))
10         for k in range(2,ulim+1):
11             if n%k==0:
12                 return False
13         return True
14
15 if __name__ == "__main__":
16     import sys
17     USAGE='python ' + sys.argv[0] + ' int'
18     if len(sys.argv) != 2:
19         raise RuntimeError(USAGE)
20     else:
21         n = int( sys.argv[1] )
22         if is_prime(n):
23             print(sys.argv[1]+' is prime.')
24         else:
25             print(sys.argv[1]+' is not prime.')
```

`sys.arv[0]` stores name of script

Checking `__main__`

Caution: when we run code in Jupyter, we are in `__main__`. So, if we run this code in Jupyter, the if-statement will execute, and try to retrieve the nonexistent command line arguments from `sys.argv`.

prime.py

```
1  import math
2
3  def is_prime(n):
4      if n <= 1:
5          return False
6      elif n==2:
7          return True
8      else:
9          ulim = math.ceil(math.sqrt(n))
10         for k in range(2,ulim+1):
11             if n%k==0:
12                 return False
13         return True
14
15 if __name__ == "__main__":
16     import sys
17     USAGE='python ' + sys.argv[0] + ' int'
18     if len(sys.argv) != 2:
19         raise RuntimeError(USAGE)
20     else:
21         n = int( sys.argv[1] )
22         if is_prime(n):
23             print(sys.argv[1]+' is prime.')
24         else:
25             print(sys.argv[1]+' is not prime.')
```