Advanced Python

Poruri Sai Rahul,
Software Developer, Enthought,
@rahulporuri

Advanced Python

- Handling files
- Loops
- Conditional statements
- The Pythonic way
- Types
- Iterators
- Functions
- Classes and Methods
- Generators

@rahulporuri

Advanced Python

- Reading from and Writing to files
- Loops, Conditionals and the Pythonic way of doing things
- Types in Python
- Iterators
- Functions
- Classes and Methods
- Generators

Handling files

```
file_obj = open(‘test_file.csv’, ‘r’)

file_contents = file_obj.read()
```

The open function returns a file object. The read method is defined on the file object and it returns the file contents as a string.

Handling files

```
file_obj = open('test_file.csv', 'r')

file_contents = file_obj.read()

lines_from_file = file_contents.split('\n')
```

Usually, each line contains distinct data in our file, so we would like separate each line

Handling files

```
file_obj = open('test_file.csv', 'r')

lines_from_file = file_obj.readlines()
```

Or, we could call the readlines method, which is also defined on the file object, that will returns a list of strings, where each of the strings is an individual line. Note that these strings have the '\n' character intact, which you would want to remove.

You would want to strip the string.

Handling files

Everything is an object in Python. Everything.

As I mentioned earlier, the open function returns an object, which has a number of attributes and methods defined on it. You can look at all of them by passing the object to dir.

e.g. `dir(file_object)`

Handling files

Everything is an object in Python. Everything.

I really meant it when I said everything. The string returned by the read method on the file object is an object. The list of strings returned by the readlines method is also an object, an object that holds references to other objects.

Think struct in C.

Handling files

- `str.strip`
- `str.split`
- `list.append`
- `list.pop`

are a few popular methods defined on the string and list objects. Look at all of them using `dir`.

Handling files

- Where is the open function defined?
- Where is the printf function defined in C?

Python *builtins*. Do `dir(__builtins__)` to get the complete list of builtins, which are imported as soon as the Python interpreter is started.

Loops

What not to do :

```
list_of_file_lines = file_obj.readlines()

no_of_lines = len(list_of_file_lines)

for i in range(no_of_lines):

    print(list_of_file_lines[i])
```

Loops

What is acceptable:

```
list_of_file_lines = file_obj.readlines()

for line in list_of_file_lines:

    print(line)
```

Loops

Best practice :

```
for line in file_obj:
    print(line)
```

Loops

Beneath the sheets, Python is using the Iterator protocol. Which can be used to iterate over containers.

- `for line in file_obj:`
- `for value in list:`
- `for key, value in dict:`
- `for item in custom_iterator:`

Conditional Statements

What not to do :

```
for i in range(no_of_lines):
    if i>0:
        print(line)
```

Advanced Python

@rahulporuri

Conditional Statements

Best practice :

```
for index, line in enumerate(file_obj):
    if i>0:
        print(line)
```

Handling files

Files can be written to using the write method on the file objects. The write method expects a string as the input, which will be written to the file. The acceptable solution is :

```
file_obj = open(‘test_file.csv’, ‘w’)

file_obj.write(string)

file_obj.close()
```

Handling files

The best practice is :

```
with open(‘test_file.csv’, ‘w’) as file_obj:
    file_obj.write(string)
```

Follow a similar practice when reading from files, which removes the need for an explicit call to the close method.

Handling files

If you are computing values that need to be stored into a file, you can use string formatting. Note that newlines (\n) need to be explicitly mentioned in the string.

```
with open(‘test_file.csv’, ‘w’) as file_obj:
    file_obj.write(“{}”.format(var))
```

Loops

What is acceptable:

```
list_of_file_lines = file_obj.readlines()

for line in list_of_file_lines:

    list_of_vars = line.split(‘,’)

    a = list_of_vars[0]

    b = list_of_vars[1]
```

Loops

What is acceptable:

```
list_of_file_lines = file_obj.readlines()

for line in list_of_file_lines:

    a, b = line.split(‘,’)
```

Loops

```
list_of_file_lines = file_obj.readlines()

for line in list_of_file_lines:

    try:

        a, b = line.split(‘,’)

    except:

        Print(“there’s something wrong with
    the string {}”.format(line))
```

Advanced Python

- Handling files
- Loops
- Conditional statements
- The Pythonic way
- Types
- Iterators
- Functions
- Classes and Methods
- Generators

@rahulporuri

Loops

```
list_of_file_lines = file_obj.readlines()

for line in list_of_file_lines:

    try:

        list_of_vars = line.split(‘,’)

    except ValueError:

        Print(“there’s something wrong with
    the string {}”.format(line))
```

The Pythonic way

Don't write C code with different syntax. Write Python code and take advantage of the Python language and it's dynamic nature.

Types

A few types that are a part of builtins are :

- `Integer`
- `String`
- `List`
- `Dictionary`

Iterators

- As we saw earlier, Iterators are containers which can be iterated over. This iteration is defined by the `__iter__` special method on the containers/objects.
- The `for` loop implicitly generates an iterator from the containers and calls `next` on the iterator inside the loop.

```
iterator = iter([1,2,3])

next(iterator) or iterator.next()
```

Advanced Python

Functions

```
def func(*args, **kwargs):

    # do_something_with_input

    return _
```

Functions

```
def custom_func(a, b):

    return (a+b)*(a-b)
```

Can be called as

- `func(1,2)`
- `func(1, b=2)`
- `func(a=1, b=2)`
- `func(a=1, 2)` - Wont work.

Advanced Python

@rahulporuri

Functions

```
def custom_func(a, b):

    return (a+b)*(a-b)
```

Can be called as

- func(*[1,2])
- func(**{‘a’:1, ‘b’:2})

Functions

```
def custom_func(a, b=None):

    return (a+b)*(a-b)
```

Can be called as

- `func(1)`
- `func(1,2)`

Functions

```
custom_func = lambda a, b: (a+b)*(a-b)
```

The call signature of lambda functions is the same as what we have seen so far. They are simply a more concise way of writing the function.

Functions

- Remember that everything is an object in Python so the function definition actually creates an object in Python.
- Try `dir(custom_func)` for fun.
- Functions are first class objects in Python - functions can be returned by other objects and functions can be passed to other objects - unlike functions in C++/Java.

Advanced Python

@rahulporuri

Classes and Methods

```
class CustomClass(object):

    def __init__(self, *args, **kwargs):

        # do something

        return None

    def custom_method(self, *args, **kwargs):

        # do something else
```

Classes and Methods

- Calling a class object generates an instance.
- Methods are simply special functions whose first argument is self, which refers to the fact that the method acts on the specific instance of a class.
- self is more of a convention that necessary Python syntax.

Generators

Generators are lazy, like Python. Calling the generator creates a generator object, which can then be iterated over.

```
def custom_generator(*args, **kwargs):

    # do something

    yield *args, **kwargs
```

Generators

The call signature is

```
generator = custom_generator(*args,
**kwargs)

next(generator)
```