

# An Introduction to Python

Parham Alvani

Amirkabir University of Technology

`parham.alvani@gmail.com`

May 15, 2015



# Keyboard Input

- ▶ The `input()` function reads a line from `sys.stdin` and returns it with the trailing newline stripped.

```
name = input("Enter your input: ")  
print("Received input is : ", name)
```

- ▶ The open Function
- ▶ The file Object
- ▶ The close() Method
- ▶ The write() Method
- ▶ The read() Method
- ▶ See [here](#) for more details and functions.

# The open Function

- ▶ Before you can read or write a file, you have to open it using Python's built-in `open()` function.
- ▶ This function creates a file object, which would be utilized to call other support methods associated with it.

---

```
file object = open(file_name [, access_mode][, buffering])
```

---

```
fo = open("foo.txt", "w")
```

---

# The write() Method

- ▶ The write() method writes any string to an open file.

```
fileObject.write(string)
```

```
# Open a file
```

```
fo = open("foo.txt", "wb")
```

```
fo.write( "Python is a great language.\nYeah its great!!\n")
```

```
# Close opened file
```

```
fo.close()
```

# The read() Method

- ▶ The read() method reads a string from an open file.

```
fileObject.read([count])
```

```
# Open a file  
fo = open("foo.txt", "r+")  
str = fo.read(10);  
print "Read String is : ", str  
# Close opened file  
fo.close()
```

# The readline() Method

- ▶ Read one entire line from the file.
- ▶ A trailing newline character is kept in the string.
- ▶ If the size argument is present and non-negative, it is a maximum byte count (including the trailing newline) and an incomplete line may be returned.

```
fileObject.readline([size])
```

# The readlines() Method

- ▶ Read until EOF using `readline()` and return a list containing the lines thus read.

```
fileObject.readlines([sizehint])
```



# Iterators & Generators

- ▶ There are many types of objects which can be used with a for loop. These are called **iterable** objects.
- ▶ The built-in function **iter** takes an **iterable** object and returns an iterator.

```
>>> x = iter([1, 2, 3])
>>> x
<listiterator object at 0x1004ca850>
>>> x.next()
1
>>> x.next()
2
>>> x.next()
3
```

# Iterators & Generators

- Iterators are implemented as classes.

```
class xrange:
    def __init__(self, n):
        self.i = 0
        self.n = n

    def __iter__(self):
        return self

    def next(self):
        if self.i < self.n:
            i = self.i
            self.i += 1
            return i
        else:
            raise StopIteration()
```

# Iterators & Generators

- In the above case, both the iterable and iterator are the same object. Notice

```
class xrange:
    def __init__(self, n):
        self.n = n

    def __iter__(self):
        return xrange_iter(self.n)
```

# Iterators & Generators

```
class xrange_iter:
    def __init__(self, n):
        self.i = 0
        self.n = n

    def __iter__(self):
        # Iterators are iterables too.
        # Adding this functions to make them so.
        return self

    def __next__(self):
        if self.i < self.n:
            i = self.i
            self.i += 1
            return i
        else:
            raise StopIteration()
```

If both iterable and iterator are the same object, it is consumed in a single iteration.

- ▶ A module allows you to logically organize your Python code.

# Questions?