Directory listing:

>>> import os

>>> entries = os.listdir('my\_directory/')

>>> entries = os.listdir('my\_directory/')

>>> for entry in entries:

print(entry)

import os

with os.scandir('my\_directory/') as entries:

for entry in entries:

print(entry.name)

Listing files in a directory:

import os

# List all files in a directory using os.listdir

basepath = 'my\_directory/'

for entry in os.listdir(basepath):

if os.path.isfile(os.path.join(basepath, entry)):

print(entry)

import os

# List all files in a directory using scandir()

basepath = 'my\_directory/'

with os.scandir(basepath) as entries:

for entry in entries:

if entry.is\_file():

print(entry.name)

Listing subdirectories:

import os

# List all subdirectories using os.listdir

basepath = 'my\_directory/'

for entry in os.listdir(basepath):

if os.path.isdir(os.path.join(basepath, entry)):

print(entry)

from pathlib import Path

# List all subdirectory using pathlib

basepath = Path('my\_directory/')

for entry in basepath.iterdir():

if entry.is\_dir():

print(entry.name)

Creating directory:

from pathlib import Path

p = Path('example\_directory')

try:

p.mkdir()

except FileExistsError as exc:

print(exc)

>>> import os

>>> # Get .txt files

>>> for f\_name in os.listdir('some\_directory'):

... if f\_name.endswith('.txt'):

... print(f\_name)

Creating temporary files:

from tempfile import TemporaryFile

# Create a temporary file and write some data to it

fp = TemporaryFile('w+t')

fp.write('Hello universe!')

# Go back to the beginning and read data from file

fp.seek(0)

data = fp.read()

# Close the file, after which it will be removed

fp.close()

Deleting files:

import os

data\_file = 'C:\\Users\\vuyisile\\Desktop\\Test\\data.txt'

os.remove(data\_file)

import os

data\_file = 'home/data.txt'

# Use exception handling

try:

os.remove(data\_file)

except OSError as e:

print(f'Error: {data\_file} : {e.strerror}')

import os

trash\_dir = 'my\_documents/bad\_dir'

try:

os.rmdir(trash\_dir)

except OSError as e:

print(f'Error: {trash\_dir} : {e.strerror}')

import shutil

src = 'path/to/file.txt'

dst = 'path/to/dest\_dir'

shutil.copy2(src, dst)

>>> import shutil

>>> shutil.move('dir\_1/', 'backup/')

'backup'

>>> from pathlib import Path

>>> data\_file = Path('data\_01.txt')

>>> data\_file.rename('data.txt')

OOP in Python:

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

if \_\_name\_\_ == "\_\_main\_\_":

p = Person("kpi", 23)

print(p.name)

class Person:

def \_\_init\_\_(self):

pass

# Single level inheritance

class Employee(Person):

def \_\_init\_\_(self):

pass

# Multi-level inheritance

class Manager(Employee):

def \_\_init\_\_(self):

pass

# Multiple Inheritance

class Enterprenaur(Person, Employee):

def \_\_init\_\_(self):

pass

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def show\_salary(self):

print("Salary is unknown")

class Employee(Person):

def \_\_init\_\_(self, name, age, salary):

super().\_\_init\_\_(name, age)

self.salary = salary

def show\_salary(self):

print("Salary is", self.salary)

if \_\_name\_\_ == "\_\_main\_\_":

p = Person("y", 23)

x = Employee("x", 20, 100000)

p.show\_salary() # Salary is unknown

x.show\_salary() # Salary is 100000

Reference: [Object Oriented Programming in Python : Learn by Examples (listendata.com)](https://www.listendata.com/2019/08/python-object-oriented-programming.html)

Regular expressions (regex) in Python:

import re

pattern = '^a...s$'

test\_string = 'abyss'

result = re.match(pattern, test\_string)

if result:

print("Search successful.")

else:

print("Search unsuccessful.")

# Program to extract numbers from a string

import re

string = 'hello 12 hi 89. Howdy 34'

pattern = '\d+'

result = re.findall(pattern, string)

print(result)

# Output: ['12', '89', '34']

import re

string = 'Twelve:12 Eighty nine:89 Nine:9.'

pattern = '\d+'

# maxsplit = 1

# split only at the first occurrence

result = re.split(pattern, string, 1)

print(result)

# Output: ['Twelve:', ' Eighty nine:89 Nine:9.']

# Program to remove all whitespaces

import re

# multiline string

string = 'abc 12\

de 23 \n f45 6'

# matches all whitespace characters

pattern = '\s+'

# empty string

replace = ''

new\_string = re.subn(pattern, replace, string)

print(new\_string)

# Output: ('abc12de23f456', 4)

**Generators in Python:**

What is a generator function?

A generator function is similar to a function in python but it gives an iterator-like generator to the caller as output  instead of an object or a value. Also, we use yield statements instead of return statements in a generator function. The yield statement pauses the execution of the generator function whenever it is executed and returns the output value to the caller. A generator function can have one or more than one yield statements but it cannot have a return statement.

### **yield vs. return**

The **yield** statement is responsible for controlling the flow of the generator function. It pauses the function execution by saving all states and yielded to the caller. Later it resumes execution when a successive function is called. We can use the multiple yield statement in the generator function.

The return statement **returns** a value and terminates the whole function and only one return statement can be used in the function.

def simple():

**for** i in range(10):

**if**(i%2==0):

         yield i

#Successive Function call using **for** loop

**for** i in simple():

    print(i)

Lambda expressions:

lambda\_func = lambda x: True if x\*\*2 >= 10 else False

lambda\_func(3) # Returns False

lambda\_func(4) # Returns True

**Comprehensions**:

With comprehensions you can construct python sequences. In this article we will see how to create and access such sequences.

Below are the different types of comprehensions in python.

* List Comprehensions
* Dictionary Comprehensions
* Set Comprehensions
* Generator Comprehensions

Day = ['Mon', 'Tue', 'Wed',]

Time= ['2pm','10am','11am']

# Create an empty dict

dictA = {}

# Use for loop

for (key, value) in zip(Day, Time):

   dictA[key] = value

print("Dictionary using for loop:\n",dictA)

listA = [12, 4, 25, 12,4,9]

genrtr = (var for var in listA if var % 2 == 0)

print("Values using generator comprehensions:\n")

for x in genrtr:

   print(x, end=',')

**Packages**:

To understand packages, you also need to know about modules. Any Python file is a module, its name being the file's base name/module's \_\_name\_\_ property without the .py extension. A package is a collection of Python modules, i.e., a package is a directory of Python modules containing an additional \_\_init\_\_.py file. The \_\_init\_\_.py distinguishes a package from a directory that just happens to contain a bunch of Python scripts. Packages can be nested to any depth, provided that the corresponding directories contain their own \_\_init\_\_.py file.

Ref: [Python Packages with Examples - Python Geeks](https://pythongeeks.org/python-packages/)