Pickle module

The pickle module implements binary protocols for serializing and de-serializing a Python object structure. *“Pickling”* is the process whereby a Python object hierarchy is converted into a byte stream, and *“unpickling”* is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as “serialization”, “marshalling,” or “flattening”; however, to avoid confusion, the terms used here are “pickling” and “unpickling”.

pickle.dump(*obj, file)*
Write the pickled representation of the object *obj* to the open file object *file*

Example
```
import pickle
def main():
    f=open("file1.ser","wb")
    pickle.dump(65,f)
    pickle.dump(1.5,f)
    pickle.dump(1+2j,f)
    pickle.dump(True,f)
    pickle.dump("PYTHON",f)
    f.close()

main()
```

Output:
The objects are converted into sequence of bytes and store these bytes inside file1.ser

pickle.load(*file)*
Read the pickled representation of an object from the open file object *file* and return the reconstituted object hierarchy specified therein

```
import pickle
def main():
    f=open("file1.ser","rb")
    a=pickle.load(f)
    b=pickle.load(f)
    c=pickle.load(f)
    d=pickle.load(f)
    e=pickle.load(f)
    print(a,b,c,d,e,sep="\n")
    f.close()

main()
```

Output:

```
65
1.5
(1+2j)
True
PYTHON
>>>
```

employee.py	**prog1.py**	**prog2.py**

class Employee:
    def __init__(self):
        self.empno=101
self.ename="suresh"
        self.salary=9000
	import pickle
	from employee import Employee
	def main():
	    emp1=Employee()
	f=open("emp.ser","wb")
	pickle.dump(emp1,f)
	    f.close()
	main()
		import pickle
		def main():
		f=open("emp.ser","rb")
		    emp1=pickle.load(f)
		print(emp1.empno,emp1.ename,emp1.salary)
		
		main()

OS Module

Operating System (OS) module is a predefined module which comes with python software.
Os module provides set of function used to communicate with operating system or to execute functions/commands of operating system.
The functions of OS module is operating system dependent
This module provides a portable way of using operating system dependent functionality

os.name

The name of the operating system dependent module imported. The following names have currently been registered: 'posix', 'nt', 'java'

posix → Unix
nt → Windows
java → Solaries

os.getcwd()
Return a string representing the current working directory.

Example:

```
import os
def main():
    res=os.getcwd()
    print(res)

main()
```

Output:

```
C:\Users\user\Desktop\python6pm
>>>
```

Application directory or the place where a program is saved and executed is called current working directory

os.chdir(*path*)
Change the current working directory to *path*.

```
import os
def main():
    os.chdir("e:\\")
    print(os.getcwd())
    f=open("p1.py","r")
    s=f.read()
    print(s)


main()
```

Output:

```
e:\
def fun1():
    print("function1")

>>>
```

os.mkdir(*path)*
Create a directory named *path* with numeric mode *mode*
If the directory already exists, FileExistsError is raised.

```
# write a program to create a directory
import os
def main():
    dname=input("enter directory name with path")
    try:
        os.mkdir(dname)
        print("directory created")
    except  FileExistsError as f:
        print("folder with this name exists")

main()
```

```
enter directory name with pathfolder1
directory created
>>>

enter directory name with pathfolder1
folder with this name exists
>>>
```