Assignment 2

What is the difference between an iterator and a generator in Python? Explain with an example.

Iterator

created using iter() function.
iterators are less memory efficient.
maintain entire iterable in memory.
not able to maintain state across iterations.
generally use for iterating through previous collections.
raises 'StopIteration' when no more items there.

Example

```
list1 = [1,2]
iterator1 = iter(list1)
print(next(iterator1))
print(next(iterator1 ))
print(next(iterator1 ))
# output is : 1 2 StopIteration
```

Generator

created using yield() function.
generators are memory efficient.
generate values only when required.
can maintain state across iterations.
ideal for creating lazy sequence or working with large data.
raises 'StopIteration' when no more items there.

Example

```
def Generator1():
for i in range (1,3):
yield i


generator1 = Generator1()


print(next(generator1 ))
print(next(generator1 ))
print(next(generator1 ))


# output is : 1 2 StopIteration
```

How does the 'yield' keyword work in Python generators? Provide an example to illustrate its usage.

working of Yield keyword

yield keyword is fundamental and necessary in generators.
yield is used to make a function as a generator function.
this generator returns one value at a time.
when generator reaches yield, it pauses, returns yielded value.
next time it remembers and resumes from where it paused.
internal state of generator is preserved between yield and next() function.

Example

```
def Generator1():
for i in range (1,3):
yield i


generator1 = Generator1()


print(next(generator1 ))
print(next(generator1 ))
print(next(generator1 ))


# output is : 1 2 StopIteration
```

Explain the use of 'os' module for file system operations in Python. Provide an example of how to create a new directory using 'os' module.

os module

os module is versatile accorss different operating systems.
os module allows to interact with operating system.
particularly for file system tasks or operations.
it allows to navigate directories, create or delete files and directories.
also permits to work with environment variables and much more.

Example

```
import os

directory_path = "osmodule_file"

# directory path is provided as per our own requirements


if not os.path.exists(directory_path):

print(f"directory {directory_path} is successfully created")

else:

print(f"sorry!! directory {directory_path} is already exists")


# output is : directory osmodule_file is successfully created
```

What are the basic exception handling keywords in Python? Provide a brief explanation of each keyword.

Exception handling

it is a mechanism that is used to handle runtime errors also known as expections.

without exception handling if any error occurs in program will stop the programm unexpectely, without any warning.
In pyton, try and except blocks are used to handle exeptions.
try block contains code that might raise errors.
except block contains code that handles these errors.

some Basic expeption handling keywords

ZeroDivisionError - Raised when second argument of division operation is zero.
TypeError - Raised when function is applied to an object of inappropriate type.
ValueError - Raised when a built-in function receives an argument that has the right type but an inappropriate value.
IndexError - Raised when a sequence subscript is out of range.
KeyError - Raised when a dictionary key is not found.
FileNotFoundError - Raised when a file or directory is requested but doesn't exist.
IOError - Raised when an I/O operation fails for an I/O-related reason.
ImportError - Raised when an import statement fails to find the module that is to be imported.
MemoryError - Raised when an operation runs out of memory.
OverflowError: - Raised when the result of an arithmetic operation is too large to be expressed by the normal number format.
AttributeError - Raised when an attribute reference or assignment fails.
SyntaxError - Raised when the parser encounters a syntax error.
IndentationError - Raised when there is incorrect indentation.
NameError - Raised when a local or global name is not found.

Write a Python code snippet to handle the 'FileNotFound' exception when opening a file using 'try-except' block.

"FileNotFoundError" exception occurs when trying to open a file that doesn't exists.

Example

```
file_name = "iNeuron.txt"

try:

with open(file_name , 'r') as file1:

file1_content = file1.read()

print(f" successful, Here is your content : {file1_content}")

except FileNotFoundError:

print(f"...sorry!! file {file_name} was not found...")


# output is : ...sorry!! file iNeuron.txt was not found...
```

Discuss the significance of the 'next' function in Python iterator. Provide a simple example to demonstrate its usage.

Significance of next function in iterator

next function gives full control over the iteration process.
it allows us to retrive next item from iterator one by one.
it raises 'StopIteration' when no more items available.
it allows to resume and pause the process by maintaining state of iterator.

Example

```
list1 = [1,2]

iterator1 = iter(list1)
```

```
print(next(iterator1))
# output is : 1

print(next(iterator1))
# output is : 2

print(next(iterator1))
# output is : StopIteration
```

What is the difference between 'os.remove()' and 'os.unlink()' functions in Python's 'os' module with regards to file system operations?

os.remove() and os.unlink()

both functions are same just with differnt names.
both do the same operation ie deleting specific file.
unlink is commonly used in unix/ linux while
remove is common in other systems like windows.
so we can use either remove or unlink to delete a file.
both raises FileNotFoundError if file doesn't exists.

Example

```
import os
file_name = "iNeuron.txt"

if os.path.exists(file_name):
```

```
  os.remove(file_name)
  #os.unlink(file_name)
  print(f" successfully removed {file_name}")

else:
  raise FileNotFoundError(f" sorry!! {file_name} was not found ")

# output is : successfully removed iNeuron.txt .....if file present
# output is : FileNotFoundError : sorry!! iNeuron.txt was not found.....if not present
```

Explain the concept of 'StopIteration' exception in Python iterators. How can it be handled?

StopIteration

fundamental mechanism for controlling iterators stop and allows better handling of iterators end.
stopiteration is used to indicate that no more items there.
next() method raises stopiteration when no items left to iterate.
in loops python automatically catches stopiteration and stops process.
manually when we use next method it also stops the process.

Example

```
list3 = [1,2]
iterator3 = iter(list3)
while True:
try:
```

```
values = next(iterator3)
print(values)
except StopIteration:
print('no more values')
break
#output is : 1 2 no more values
```

Write a Python code snippet to iterate through a file line by line using file handling and a generator.

```
file_name = 'iNeuron1.txt'

def read_lines(file_name):
with open(file_name, 'r') as file5:
for line in file5:
yield line.strip()
for line in read_lines(file_name):
print(line)
```

Discuss the role of 'finally' block in Python exception handling. Provide a scenario where 'finally' block would be useful.

Role of finally block

code inside finally block always executed regardless of exceptions.

it is typically used with try block and sometimes with except block.
it is used for cleanup task, file handling, and network connections.
it can contain important cleanup code to prevent resource leaks.
it also ensures data security and integrity.
finally block is a key part of writing robust and reliable code.

Example

```
file_name = 'iNeuron1.txt'
try:
with open(file_name, 'r') as file6:
file6_contert = file6.read()
print('successful, here is your data : ', file6_contert )
except:
raise FileNotFoundError(f" sorry!! file was not found ")
finally:
file6.close()
print('file is closed now')
```

Explain the purpose of 'os.walk()' function in Python's 'os' module. Provide an example demonstrating its usage.

os.walk() function

used to generated file names in directory tree.
walking in trees is done either by top-down or bottom-up.
very essential to traverse between directory structure.

can be used to list all files an directories frequently.
when os.walk() is used it returns a generator that yields a tuple for each directory it traverse through.
this allows to inspect files and directories without writing complex code.

Example

```
import os

root_directory = "c:/users/"

for root, dirs, files in os.walk(root_directory):

print('current directory', root)

print('subdirectories', dirs)

print('files', files)

# we can also check full path further
```

What is the 'raise' keyword used for in Python? How is it related to exception handling?

raise keword

raise keyword is used to throw exceptions in pyhton.
it prevents from unexpected crash and allows better exception handling.
it uses try, except, finally, else block to handle exceptions effectively.
we can raise built-in exceptions as well as custom exceptions.

Example

```
x = 2

if x < 10:
```

```
raise Exception("There is an issue, x is below zero")

# output is : Exception: There is an issue, x is below zero
```

Write a Python code snippet to create a new file and write data into it using file handling and exception handling for file I/O errors.

```
file_name = 'iNeuron2.txt'

try:

with open (file_name, 'w')as file7:

file7.write('this is first and last line')

print('writing successful')

except FileNotFoundError:

print(" sorry!! file was not found ")

except IOError:

print("oops!! I/O error occured")

except Exception as e:

print(" unexpected error ")
```