Programming Fundamentals and Python

Recursion

Recursion

Python also accepts function recursion, which means a defined function can call itself.

Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.

Recursion

CODE

```
def tri_recursion(k):

  if(k > 0):
    result = k + tri_recursion(k - 1)
    print(result)
  else:
    result = 0
  return result

print("\n\nRecursion Example Results")
tri_recursion(6)
```

OUTPUT

```
Recursion Example Results
1
3
6
10
15
21
>>>
```

Recursion

A recursive function is a function that calls itself.
The idea is to represent a problem in terms of one or more smaller problems.

<u>Components / properties of a recursive function:</u>

Base Case
- Indicates the stopping condition.
- Could be more than one.

Recursive Call
- Moves the execution towards the base case.
- Could be more than one.

Recursion

A recursive function is a function that calls itself.
The idea is to represent a problem in terms of one or more smaller problems.

<u>Components / properties of a recursive function:</u>

Base Case
- Indicates the stopping condition.
- Could be more than one.

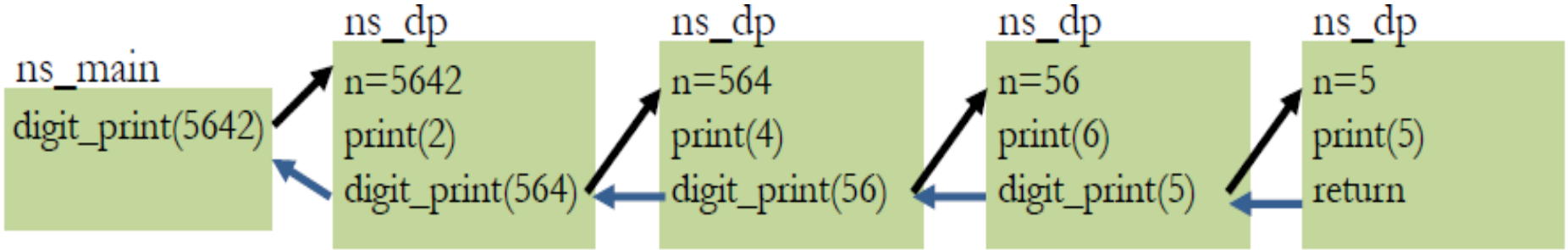
Recursive Call
- Moves the execution towards the base case.
- Could be more than one.

Recursion with a Base Case

Example: Write a recursive function that prints a decimal number digit-wise, starting from the lowest order digit.

```
def digit_print(n):
    print(n%10)
    if n<10:
        return
    digit_print(n//10)
digit_print(5642)
```

```
>>>
==== RESTAR
ython38/1.p
2
4
6
5
>>>
```



Why Use Recursion??

- Recursion is made for solving problems that can be broken down into smaller, repetitive problems.
- It is especially good for working on things that have many possible branches and are too complex for an iterative approach.
- Recursion may be efficient from programmer's point of view.
- Recursion may not be efficient from computer's point of view.

Functions as Objects

- Everything in Python is an object.
- Functions are objects too!

- have types
- can be assigned to names
- can be used in expressions
- can be passed as arguments to other functions
- stored in various data structures like lists and dictionaries
- a function can be defined within another function – nested function

Files

Files

A file is a sequence of bytes stored on a secondary memory device.

Files are of various types:

- Text Files
- Spreadsheets
- Binary Files
- Executable Files

Files are managed by File system that operating system supports.

Processing a file is based on three steps:

- Operating a file for reading or writing
- Reading from or writing to the file
- Closing the file

Opening a file

The function open() is used to open the files (text or binary). This function is defined in built-in modules.

The function takes:

- A file name (with or without path)
- '\' is used for path but since it may coincide with escape sequence so python accepts '/' (forward slash).
- A path could be absolute or relative:

  - Raw / absolute path starts from root directory:

  e.g.: c:\office\classes\CP\Text.txt

  - Relative path starts the sequence from current directory:

  e.g: CP\Text.txt

Opening a file

The function takes:

- A file name (with or without path).
- Mode specifies how to interact with opened file.
- r=>reading mode (default)
- w=>writing mode, if the file already exists otherwise its content is wiped out
- a=>append mode, the data will append to the end of file.
- t=>text mode (default)
- b=>binary mode

Opening a file


```
>>> f=open('myfile.txt') ≡ f=open('myfile.txt', 'r')
```

- Opens myfile.txt if it exist
- Generates error if the file does not exist.
- The file is opened for reading only.

```
>>> f=open('myfile.txt', 'w')
```

- Opens myfile.txt for writing.
- Creates a new file if it doesn't exist.
- Overwrites the existing file.

Opening a file


```
>>> f=open('myfile.txt', 'a')
```

- Opens myfile.txt for writing.
- Creates a new file if it doesn't exist.
- Appends at the end of existing file.

with/as statement

- Automatically closes a file after block of code is executed.

Syntax:

```
with open <file name> as f:
    <block>
```

- Opens <file name> and assigns it handler.
- Closes f after <block> is executed.

Reading a file

f.read(n)

- reads and returns as string ‘n’ characters from file
- ‘f’ or until the end of file is reached.

f.read()

- reads and returns as string characters from file f until the end of file.

f.readline()

- reads and returns as string characters from file f until (including) new line character or end of file.

f.readlines()

- reads and returns as list.

Reading a file

- With every opened file, the system will associate a cursor that points to the character in the file.
- When the file is first opened, the cursor typically points to the start of the file.
- Using different types of read operations consecutively, second read commences from where first read ended.

Other useful Functions on files

- f.name

  Contains name of file. It's an attribute, not a method.
- f.seek(offset, from_what)

  Changes file object position.

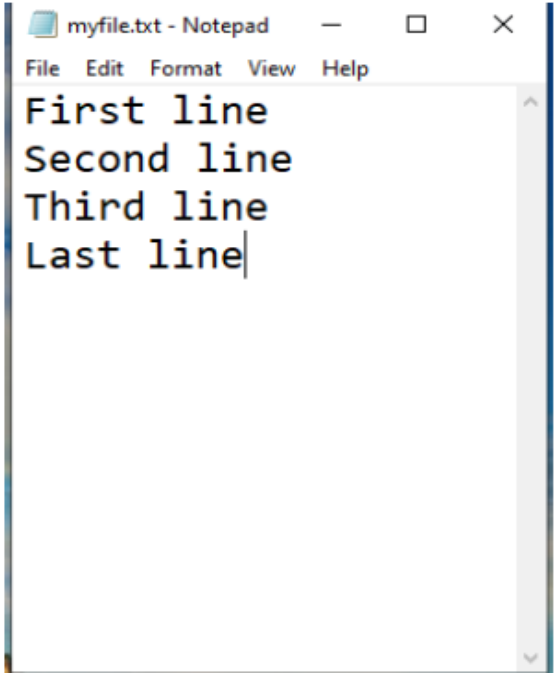
  Position is computed from adding offset to a reference point

  Reference point is selected by from_what argument.

  - From_what=0, offset measured from start of file.
  - From_what=1, offset measured from current position.
  - From_what=2, offset measured from EoF
  - Default value is 0
- f.tell(): returns an integer giving file objects current position in the file as number of bytes from the beginning of the file.

Example -1

Store the following file as myfile.text




```
>>> f=open('myfile.txt')
>>> f.read()
'First line\nSecond line\nThird line\nLast line'
>>> f.read()
''
>>> f.close()
>>> f=open('myfile.txt')
>>> print(f.read())
First line
Second line
Third line
Last line
>>> f.close()
```

Example -2

Reading one line at a time

File Edit Format View Help

```
First line
Second line
Third line
Last line
```

```
>>> f=open('myfile.txt')
>>> f.readline()
'First line\n'
>>> f.readline()
'Second line\n'
>>> f.readline()
'Third line\n'
>>> f.close()
```

Example - 3

Reading all lines

```
>>> f=open('myfile.txt')
>>> f.readlines()
['First line\n', 'Second line\n', 'Third line\n', 'Last line']
>>> f=open('myfile.txt')
>>> p=f.readlines()
>>> print(p[1])
Second line

>>> f.close()
```

Example - 4


```
>>> f=open('myfile.txt')
>>> f.read()
'First line\nSecond line\nThird line\nLast line'
>>> f.read()
''
>>> f.seek(0)
0
>>> f.read()
'First line\nSecond line\nThird line\nLast line'
>>> f.seek(3)
3
>>> f.read(5)
'st li'
>>> f.tell()
8
```