SNS COLLEGE OF ENGINEERING
Kurumbapalayam (Po), Coimbatore – 641 107

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

19IT103 – COMPUTATIONAL THINKING AND PYTHON PROGRAMMING

- **A readable, dynamic, pleasant, flexible, fast and powerful language**

Objective

Files and exception: text files, reading and writing files, format operator; command line arguments, errors and exceptions, handling exceptions, modules, packages; Illustrative programs: word count, copy file, Voter's age validation, Marks range validation (0-100).

.

RECAP

- A file is a container in a computer system for storing information.
- **Binary file :** Binary file is a collection of bytes or a character stream.
- **Text file :** A text file is a stream of characters that can be processed sequentially and logically in the forward direction.
- Creation of a new file
- Modification of data or file attributes
- Reading of data from the file
- Opening the file in order to make the contents available to other programs
- Writing data to the file
- Closing or terminating a file operation

Read a file in python

Code example

```
fileToRead = open("source.txt","r")
data = fileToRead.read()
print("Source contents:",data)
```

Function opens a file

mode

location

Method reads the file content

File contents

```
Apple Orange Mango
```

Output

```
Source contents: Apple Orange Mango
```

https://repl.it/@kiteit/ReadFromFileIntro

Writing File

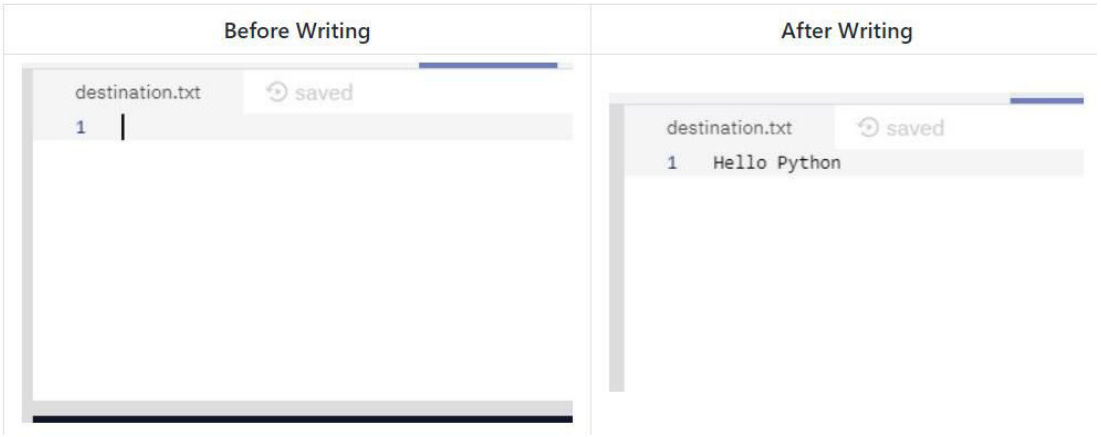
code

```
fileToWrite = open("destination.txt","w")
data = "Hello Python";
fileToWrite.write(data)
fileToWrite.close();
print("done with file writing")
```

Method writes content to the file

output

```
done with file writing
```



https://repl.it/@kiteit/WritingToFile

Read (Limited to number of characters)

SOURCE FILE

```
Line 1
Line 2
Line 3
```

CODE

```
fileToRead = open("source.txt","r")
data = fileToRead.read(3)
print(data)
```

read(no of chars)

OUTPUT

```
Lin
```

https://repl.it/@kiteit/ReadFromFile

Reading a line from file

- Method: file.readline()
- It will return a line of string every time you called.
- Default delimiter is new line character (‘\n’)

readline

https://repl.it/@kiteit/ReadLineFromFile

kiteit / ReadLineFromFile

Run

main.py source.txt

```
fileToRead = open("source.txt","r")
data = fileToRead.readline()
print(data)
```

https://ReadLineFromFile.kiteit.repl.run

```
Python 3.8.2 (default, Feb 26 2020,
```

To read all lines

source.txt saved

```
1 Line 1
2 Line 2
3 Line 3
```

https://ReadLineFromFile.kiteit.repl.run

main.py saving...

```
1 fileToRead = open("source.txt","r")
2 data = fileToRead.readline()
3 print(data)
```

https://ReadLineFromFile.kiteit.repl.run

```
Line 1
```

main.py saved

```
1 fileToRead = open("source.txt","r")
2 data = fileToRead.readline()
3 print(data)
4 data = fileToRead.readline()
5 print(data)
```

https://ReadLineFromFile.kiteit.repl.run

```
Line 1

Line 2
```

main.py saving...

```
1 fileToRead = open("source.txt","r")
2 data = fileToRead.readline()
3 print(data)
4 data = fileToRead.readline()
5 print(data)
6 data = fileToRead.readline()
7 print(data)
8
```

https://ReadLineFromFile.kiteit.repl.run

```
Line 1

Line 2

Line 3
```

To Think

- We need to perform readline operation for every line of that the sources file has.
- What if number of readline operations is lower than number of lines in the source file?
- What if we over run it? no of readline operations > no of lines in source file.
- How do we know the no of lines in the file?
- how do we stop readline operation after reaching the end of file?
- What about the idea of using loops?

Format Operator

Python uses C-style string formatting to create new, formatted strings. The "%" operator is used to format a set of variables enclosed in a "tuple" (a fixed size list), together with a format string, which contains normal text together with "argument specifiers", special symbols like "%s" and "%d".

Format Operator

Format operators

main.py

```
1   print("%s per kg %d Rs" % ("Apple", 80))
2
```

https://FormatOperator.kiteit.repl.run

```
Apple per kg 80 Rs
>
```

Format Operators

```
%s - String (or any object with a string representation, like numbers)

%d - Integers

%f - Floating point numbers

%.<number of digits>f - Floating point numbers with a fixed amount of digits
to the right of the dot.

%x/%X - Integers in hex representation (lowercase/uppercase)
```

Using format method

- The `format()` method formats the specified value(s) and insert them inside the string's placeholder.
- The placeholder is defined using curly brackets: `{}`. Read more about the placeholders in the Placeholder section below.
- The `format()` method returns the formatted string.

Format method

main.py

```
1  print("{} per kg {} Rs".format("Apple", 80))
2
```

https://FormatOperator.kiteit.repl.run

```
Apple per kg 80 Rs
```

Format method (value mapping with keyword)

main.py

```
print("{fruit} per kg {price} Rs".format(fruit="Apple", price=80))
```

https://FormatOperator.kiteit.repl.run

```
Apple per kg 80 Rs
>
```

main.py

```
print("{fruit} per kg {price} Rs".format(price=80, fruit="Apple"))
```

https://FormatOperator.kiteit.repl.run

```
Apple per kg 80 Rs
>
```

Index is not problem here

Format method (value mapping with argument index)

main.py

```
1  print("{0} per kg {1} Rs".format("Apple", 80))
2
3
```

https://FormatOperator.kiteit.repl.run

```
Apple per kg 80 Rs
```

main.py

```
1  print("{1} per kg {0} Rs".format("Apple", 80))
2
3
```

https://FormatOperator.kiteit.repl.run

```
80 per kg Apple Rs
```

Wrong Mapping

Format method (value mapping with keyword)

main.py

```
print("{fruit} per kg {price} Rs".format(fruit="Apple", price=80))
```

https://FormatOperator.kiteit.repl.run

```
Apple per kg 80 Rs
>
```

main.py

```
print("{fruit} per kg {price} Rs".format(price=80, fruit="Apple"))
```

https://FormatOperator.kiteit.repl.run

```
Apple per kg 80 Rs
>
```

Index is not problem here

Other options

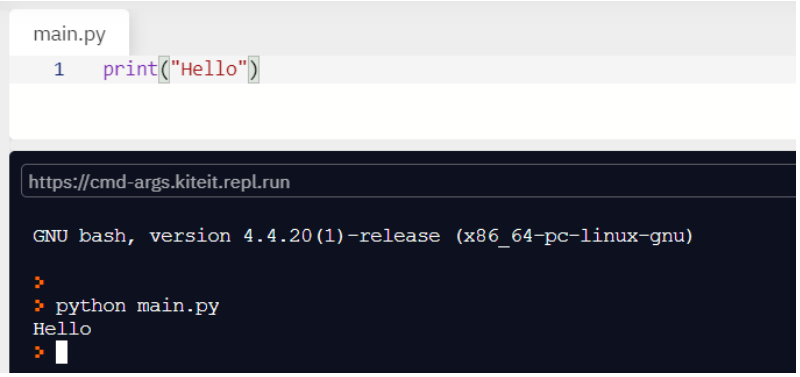
:<	Left aligns the result (within the available space)
:>	Right aligns the result (within the available space)
:^	Center aligns the result (within the available space)
:=	Places the sign to the left most position
:+	Use a plus sign to indicate if the result is positive or negative
:-	Use a minus sign for negative values only
:	Use a space to insert an extra space before positive numbers (and a minus sign before negative numbers)
:,	Use a comma as a thousand separator
:_	Use a underscore as a thousand separator
:b	Binary format
:c	Converts the value into the corresponding unicode character

Other options

:d	Decimal format
:e	Scientific format, with a lower case e
:E	Scientific format, with an upper case E
:f	Fix point number format
:F	Fix point number format, in uppercase format (show inf and nan as INF and NAN)
:g	General format
:G	General format (using a upper case E for scientific notations)
:o	Octal format
:x	Hex format, lower case
:X	Hex format, upper case
:n	Number format
:%	Percentage format

Command Line Arguments

- Command-line arguments are a common way to parameterize execution of programs.
- We can pass the parameters while running the program.
- sys is module that helps to parse the arguments



Running a python program from cmd line

Command Line Arguments

main.py

```
import sys
cmdargs=sys.argv;
print("cmd args:",cmdargs)
```

https://cmd-args.kiteit.repl.run

```
> python main.py
cmd args: ['main.py']
>
```

First argument is always a program itself

main.py

```
import sys
cmdargs=sys.argv;
print("cmd args:",cmdargs)
```

https://cmd-args.kiteit.repl.run

```
> python main.py cmd1 cmd2 cmd3
cmd args: ['main.py', 'cmd1', 'cmd2', 'cmd3']
>
```

Other [arguments to the program]c

Using command Line Arguments

main.py

```python
import sys
cmdargs=sys.argv;
value1 = int(sys.argv[1])
value2 = int(sys.argv[2])
result = value1 + value2;
print("result:",result)
```

https://cmd-args.kiteit.repl.run

```
> python main.py 2 8
result: 10
>
```

Don't forget it will be received as string (need to convert if you want)

SUMMARY

- Python uses C-style string formatting to create new, formatted strings. The "%" operator is used to format a set of variables enclosed in a "tuple" (a fixed size list), together with a format string, which contains normal text together with "argument specifiers", special symbols like "%s" and "%d".
- Command-line arguments are a common way to parameterize execution of programs.
- We can pass the parameters while running the program.
- sys is module that helps to parse the arguments

Thank you