## Passing Argument

Be careful from this common mistake :-

from dateTime import datetime

```
Def hello1 (Curdate = datetime.now()):
        Print (C) → it will Return same value
```

```
Def hello2 (Curdate = None):
    if Curdate is None:
        Curdate = datetime.now()
        Print (Curdate) ——→ For i in Range (10):
```

---

```
def Hello (lst = []):
    lst.append(1)
    Print (lst)
```

hello() → [1]
hello() → [1,1]
hello() → [1,1,1]

ended this part of assigning a mutable variable in a parameter o
Functio

↳ so every time we append to the list we get new variable valu

---

Time module → with Time module we can do several Things.
↳ Get local Time and it's Date / Time Component.
Compute Time difference Between Two Functions.

// The Computing Time is
From start Point Ca

⇒ Epoch is where Time start. Epoch → 1, 1, 1970

## Time 2

Function

Time Give real world - Time (kind of) Relative to epoch

→ Good we can understand intuitively → help

→ Maintain by dedicated h-w

Python has several other paths

→ Clock . Perf_Counter, monotonic

↳ measure Time interval , Timeit module

Provide relative Time and has no reference time Point

To calc avg of how much code Run

⇒ Perf_Counter = Time.time()

↳ Return exact Nanosecond

→ Curre

.. → Pa

Code I

Line

Cho

# Passing Argument

Be careful from this common mistake :-

```
from dateTime import datetime

Def hello1 (Curdate = datetime.now()):
        Print (C) → it will Return same value

Def hello2 (Curdate = None):
        if Curdate is None:
            Curdate = datetime.now()
            Print (Curdate) ——→ For i in Range (10):
```

---

```
def Hello (lst = []):
        lst.append(1)
        Print (lst)
```

hello() → [1]
hello() → [1,1]
hello() → [1,1,1]

ended this part of assigning a mutable variable in a parameter of function

↳ so every time we append to the list we get new variable value.

---

Time module → With Time module we can do several Things.

↳ Get local Time and it's Date / Time Component.

Compute Time difference Between Two Functions. // The Computing Time is relative

From start Point Called

⇒ Epoch is where Time start. Epoch → 1, 1, 1970

Time 2

Time function → Give real world - Time (kind of) Relative to epoch
→ Good we can understand intuitively → lap
→ Maintain by dedicated h.w

Python has several Other Paths
→ Clock . Perf_Counter, Monotonic. → measure Time interval , Timeit module
provide relative Time and has no reference time Point. To calc avg of how much code Run
⇒ Perf_Counter = Time.time() → Return exact Nanosecond

---

(text ← Files → Binary) ← Files ← most important Part

⇒ Binary File (File size smaller) and text File → Can open it when there is a corruption.
Allow text and binary Data
Store data as Sequance of bytes
Normaly are Natural text
minor error and you can't open it again.

File name, File Path → Determine Where is the File / The Type of the File
it must have extension

File Path → Relative (Doesn't start from Root) , absolute Path (start from Root.)

We can use ./ Workspace → Current W.S → We want workspace file

. → Current dir
.. → Parent dir

→ every ../ → means go up One layer and Cancel Ne.

Cancel this

Code / CPP_Skills / 2 / .. / CPP_Skills

Note: Becareful with Paths that contain Spaces.

Cancel this

Line ending → In → line Feed → Unix & linux Recognize end of line by: \n

Carratige Return → \r OR CR

Note: Windows determine end of line by \r\n

Character encoding → Common encoding → UTF-8 : Very Popular

UTF-16 : less used in Practise

(Uni code) → Cover all major living languages.

→ Repertoire of chars by some kind of encoding system that assign a number to each Character.

Open file with Lower encoding Capabilities → you will Face Problem.

Ex: File written with alot of math eq and alot of Symbols in Russian long If it's opened in ASCII it will Fail → Because it has 128 char only.

in Python it's Platform dependent → if you want to use UTF-8 → Pass it as Parameter → if you want to use OR open UTF-8 file encoded → use UTF-8 decoder.

# Reading from Files ⟶ How?

1) Reading, open, read and close:-

Code:

Path = "data.txt" ⟶ Path is valid

File = open(Path, 'r') ⟶ 'r' → mode for Reading

For line in file:

print(line, end="") ⟶ "even without /n it will be printed with a new line every time" close

file.close() ⟶ We must close every file we open.

---

Read line and Read lines method

⇒ Continue over the last code:

String = file.readline() ⟶ Read only first line

Print(String) ⟶ #1st line only

Lines = file.Readlines() ⟶ "Return a list of strings, each element is a line in the file"

---

In Pythonic Way ⟶ We use With To prevent errors from closing files and exc

→ Lines = []

open Path ⟶ and Put it in object.

```
with open(Path, 'r') as file:
    lines = file.readlines()
```

Print(lines) ⟶ Same as last but it closes automaticly

Lines = File.Read().Splitlines()

Work on it → Return a list of lines in the string, breaking at boundaries

Return whole file as string

## Encoding : UTF-8

⇒ Code:

```
Path = 'data_UTF8.txt'
With open(Path, 'r', encoding='UTF-8') as file:
    Lines = File.read().splitlines()
    Print(Lines)
With open(Path, 'r', encoding='ASCII') as file:
```

use it to read binary content

open file as object

we can't read a binary with ASCII

Because it has limit → (128) Range.

## Writing to Files

⇒ write method doesn't add new line

⇒ if File doesn't exist, it will be created → invalid if invalid Path or Security Permission.

⇒ By default old content will be overwritten.

Code:

```
Path = 'output1.txt'
With open(Path, 'w') as file:
    File.write('Hey')
    File.write('your name?')
```

→ Returned as (Heyyour name?)

**Printing Lines :-**

```
Path = 'Output2.txt'
lines = ['Hey', 'Your name!']
with open(Path, 'w') as file:  # Write over obj file
    for line in lines:
        file.write(line + '\n')
```

→ Hey
Your Name

**Appending Mode :-**

Keep adding lines to end of file

```
with open(Path, 'a') as file:
    for line in lines:
        file.write(line + '\n')
```

**Read and Write**

→ We can mix using r+ or w+ → but it's Not Recommended

We can Read and Write at Same Time.

```
in_P = 'InPut Path.txt'
out_P = 'Output Path.txt'
with open(in_P, 'w') as writer, open(out_P, 'w') as Reader:
    lines = reader.readlines()
    writer.writelines(reversed(lines))
```

↓ doesn't add In