

# **COL1000**

# **Introduction to Programming**

**Priyanka Golia**

Most (if not all) of the content is borrowed from Prof. Subodh Kumar's slides

```
Name, Dept, Marks  
x, CSE, 70  
y, MT, 75  
z, TT, 80  
a, CSE, 85
```

<marks.csv>

To calculate sum of marks obtained by CSE students?

Name, Dept, Marks  
x, CSE, 70  
y, MT, 75  
z, TT, 80  
a, CSE, 85

<marks.csv>

To calculate sum of marks obtained by CSE students?

```
1  def read(file):
2      try:
3          with open(file, "r") as f:
4              content = f.read()
5              lines = content.split("\n")
6              sum = 0
7              for line in lines:
8                  words = line.split(",")
9                  dept = words[1].strip(" ").lower()
10                 if dept == "cse":
11                     sum += int(words[2])
12             return sum
13     except FileNotFoundError:
14         print("file not found")
15     except PermissionError:
16         print("permission is not there")
17     except Exception as e:
18         print("I am in exception, ",e)
19
20 file = "../files/marks.csv"
21 sum = read(file)
22 print(f"sum of marks of cse students is {sum}")
```

# Command Line Arguments

*Why* command-line arguments are useful:

- Programs often need input (e.g., filenames, numbers, or options).
- Instead of prompting interactively, arguments let you run scripts with parameters directly.

“Imagine writing a script to add two numbers. Instead of typing them after running the program, we can provide them right when we start the program.”

```
python add.py 5 7
```

# Command Line Arguments

```
import sys  
print(sys.argv)
```



Output

```
['demo.py', 'hello', 'world', '123']
```

```
~/ $ python demo.py hello world 123
```

Each space-separated token becomes an element in the list.

`sys.argv[0]` → script name.

Everything after that → actual arguments.

You can check the length of arguments — `len(sys.argv)`

# Command Line Arguments

```
1 import sys
2
3 def greet(name):
4     print('hello! ', name)
5
6 name = sys.argv[1]
7 greet(name)
```

Output



```
[priyanka@Priyankas-MacBook-Pro col1000-lect % python3 lec32_2.py priyanka
hello!  priyanka
[priyanka@Priyankas-MacBook-Pro col1000-lect % python3 lec32_2.py sayan
hello!  sayan
[priyanka@Priyankas-MacBook-Pro col1000-lect % python3 lec32_2.py svs
hello!  svs
[priyanka@Priyankas-MacBook-Pro col1000-lect % python3 lec32_2.py subodh
hello!  subodh
[priyanka@Priyankas-MacBook-Pro col1000-lect %
```



# Command Line Arguments

```
1 import sys
2
3 def greet(name):
4     print('hello! ', name)
5
6 name = sys.argv[1]
7 greet(name)
```

what happens if you forget to give a name?

Output



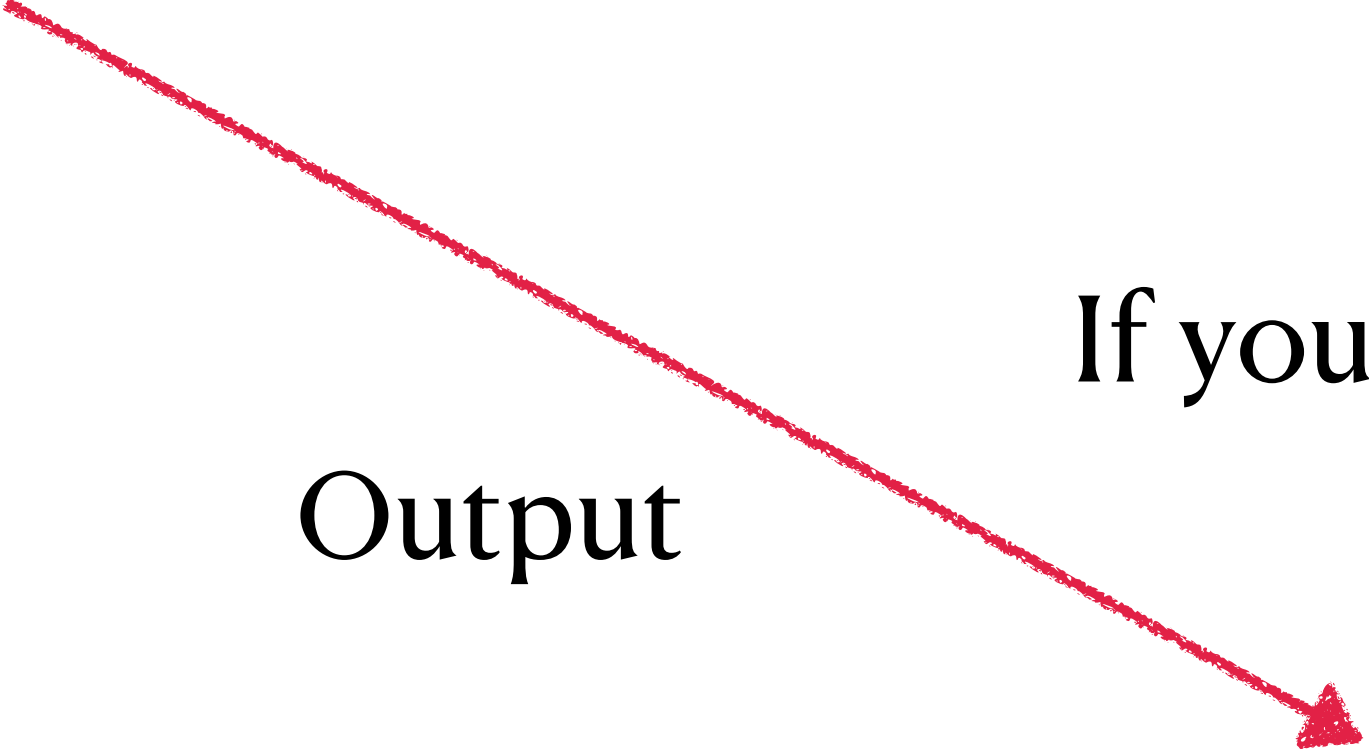
```
[priyanka@Priyankas-MacBook-Pro col1000-lect % python3 lec32_2.py priyanka
hello!  priyanka
[priyanka@Priyankas-MacBook-Pro col1000-lect % python3 lec32_2.py sayan
hello!  sayan
[priyanka@Priyankas-MacBook-Pro col1000-lect % python3 lec32_2.py svb
hello!  svb
[priyanka@Priyankas-MacBook-Pro col1000-lect % python3 lec32_2.py subodh
hello!  subodh
[priyanka@Priyankas-MacBook-Pro col1000-lect %
```

# Command Line Arguments

```
1 import sys
2
3 def greet(name):
4     print('hello! ', name)
5
6 name = sys.argv[1]
7 greet(name)
```

If you forget to give a name!

Output

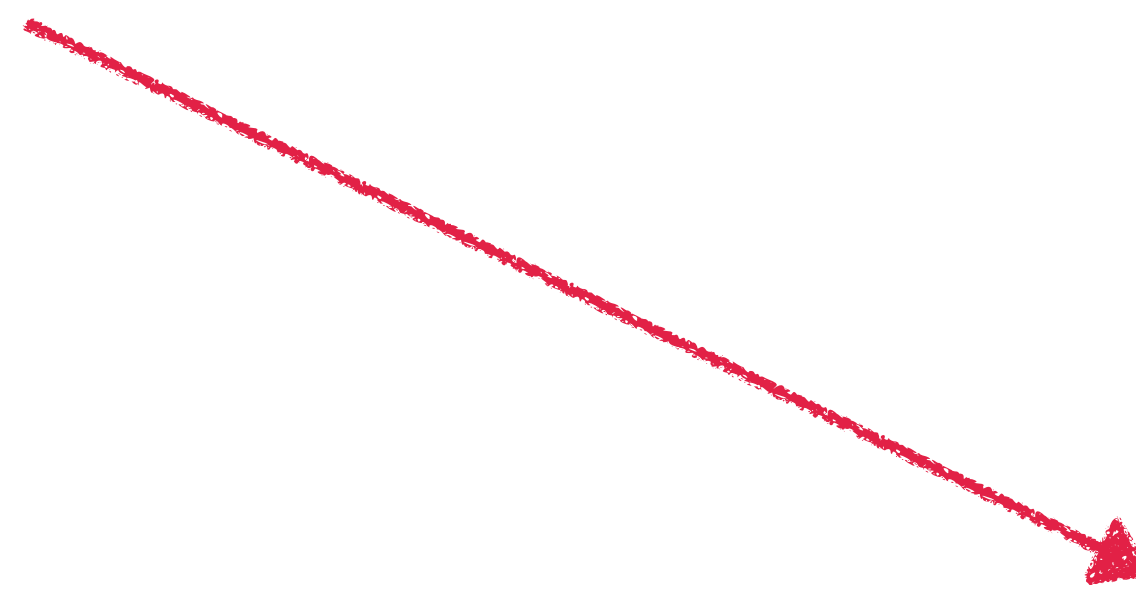


```
priyanka@Priyankas-MacBook-Pro col1000-lect % python3 lec32_2.py
Traceback (most recent call last):
  File "/Users/priyanka/Desktop/col1000-lect/lec32_2.py", line 6, in <module>
    name = sys.argv[1]
             ~~~~~^~^~
IndexError: list index out of range
```



# Command Line Arguments

```
1  import sys
2
3  def greet(name):
4      print('hello! ', name)
5
6  if len(sys.argv) == 2:
7      name = sys.argv[1]
8  else:
9      print("considering default")
10     name = "priyanka"
11 greet(name)
```



```
priyanka@Priyankas-MacBook-Pro col1000-lect % python3 lec32_2.py svsv
hello! svsv
priyanka@Priyankas-MacBook-Pro col1000-lect % python3 lec32_2.py
considering default
hello! priyanka
priyanka@Priyankas-MacBook-Pro col1000-lect %
```