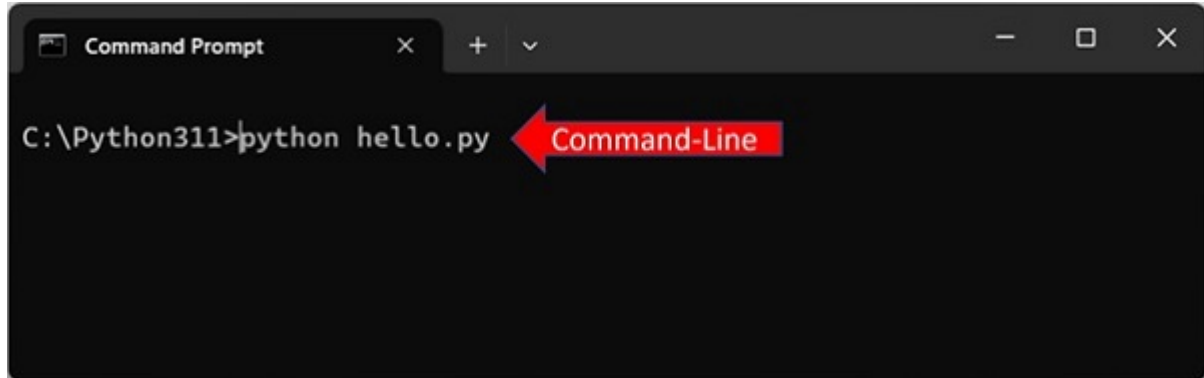


# Python - argumentos de linha de comando

Para executar um programa Python, executamos o seguinte comando no terminal de prompt de comando do sistema operacional. Por exemplo, no Windows, o seguinte comando é inserido no terminal do prompt de comando do Windows.



```
Command Prompt
C:\Python311>python hello.py
```

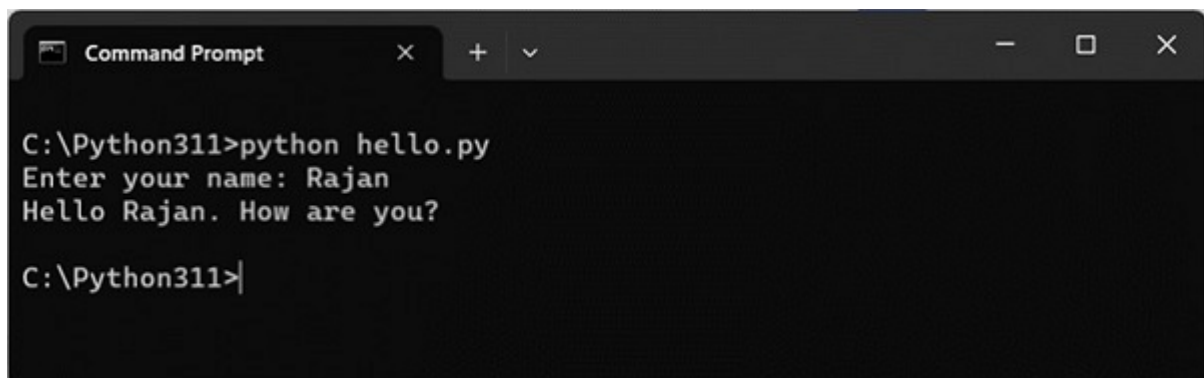
A linha na frente do prompt de comando C:\> (ou \$ no caso do sistema operacional Linux) é chamada de linha de comando.

Se o programa precisar aceitar a entrada do usuário, a função `input()` do Python será usada. Quando o programa é executado na linha de comando, a entrada do usuário é aceita no terminal de comando.

## Exemplo

```
name = input("Enter your name: ")
print ("Hello {}. How are you?".format(name))
```

O programa é executado a partir do terminal do prompt de comando da seguinte forma -



```
Command Prompt
C:\Python311>python hello.py
Enter your name: Rajan
Hello Rajan. How are you?
C:\Python311>
```

Muitas vezes, pode ser necessário colocar os dados a serem utilizados pelo programa na própria linha de comando e utilizá-los dentro do programa. Um exemplo de fornecimento de dados na linha de comando poderia ser qualquer comando DOS no Windows ou Linux.

No Windows, você usa o seguinte comando DOS para renomear um arquivo `hello.py` para `hi.py`.



```
C:\Python311>ren hello.py hi.py
```

No Linux você pode usar o comando mv -

```
$ mv hello.py hi.py
```

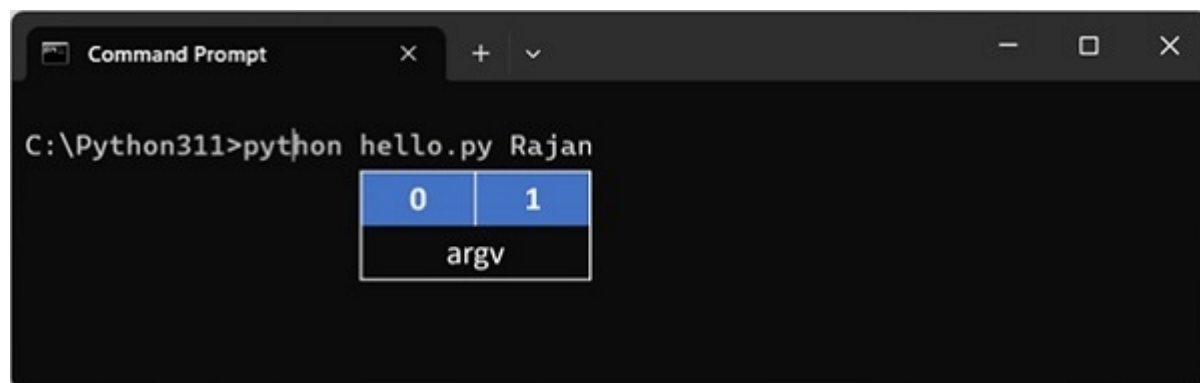
Aqui, ren ou mv são os comandos que precisam dos nomes de arquivos novos e antigos. Como são alinhados com o comando, são chamados de argumentos de linha de comando.

Você pode passar valores para um programa Python na linha de comando. Python coleta os argumentos em um objeto de lista. O módulo sys do Python fornece acesso a quaisquer argumentos de linha de comando por meio da variável sys.argv. sys.argv é a lista de argumentos da linha de comando e sys.argv[0] é o programa, ou seja, o nome do script.

O script hello.py usou a função input() para aceitar a entrada do usuário após a execução do script. Vamos alterá-lo para aceitar entradas da linha de comando.

```
import sys
print ('argument list', sys.argv)
name = sys.argv[1]
print ("Hello {}. How are you?".format(name))
```

Execute o programa na linha de comando conforme mostrado na figura a seguir -



A **saída** é mostrada abaixo -

```
C:\Python311>python hello.py Rajan
argument list ['hello.py', 'Rajan']
Hello Rajan. How are you?
```

Os argumentos da linha de comando são sempre armazenados em variáveis de string. Para usá-los como numéricos, você pode usá-los adequadamente com funções de conversão de tipo.

No exemplo a seguir, dois números são inseridos como argumentos de linha de comando. Dentro do programa, usamos a função int() para analisá-los como variáveis inteiras.

```
import sys
print ('argument list', sys.argv)
first = int(sys.argv[1])
second = int(sys.argv[2])
print ("sum = {}".format(first+second))
```

Ele produzirá a seguinte **saída** -

```
C:\Python311>python hello.py 10 20
argument list ['hello.py', '10', '20']
sum = 30
```

A biblioteca padrão do Python inclui alguns módulos úteis para analisar argumentos e opções de linha de comando -

- **getopt** - Analisador estilo C para opções de linha de comando.
- **argparse** - Analisador de opções de linha de comando, argumentos e subcomandos.

## O módulo getopt

Python fornece um módulo **getopt** que ajuda a analisar opções e argumentos de linha de comando. Este módulo fornece duas funções e uma exceção para permitir a análise de argumentos de linha de comando.

### Método getopt.getopt

Este método analisa as opções da linha de comando e a lista de parâmetros. A seguir está uma sintaxe simples para este método -

```
getopt.getopt(args, options, [long_options])
```

Aqui estão os detalhes dos parâmetros -

- **args** - Esta é a lista de argumentos a ser analisada.
- **options** - Esta é a sequência de letras de opção que o script deseja reconhecer, com opções que requerem um argumento que devem ser seguidas por dois pontos (:).
- **long\_options** - Este é um parâmetro opcional e se especificado, deve ser uma lista de strings com os nomes das opções longas, que devem ser suportadas. Opções longas, que requerem um argumento, devem ser seguidas por um sinal de igual ('='). Para aceitar apenas opções longas, as opções devem ser uma sequência vazia.

Este método retorna um valor que consiste em dois elementos - o primeiro é uma lista de pares (opção, valor), o segundo é uma lista de argumentos do programa restantes após a

remoção da lista de opções.

Cada par opção-valor retornado tem a opção como seu primeiro elemento, prefixado com um hífen para opções curtas (por exemplo, '-x') ou dois hifens para opções longas (por exemplo, '--long-option').

## Exceção getopt.GetoptError

Isso é gerado quando uma opção não reconhecida é encontrada na lista de argumentos ou quando uma opção que requer um argumento não recebe nenhum.

O argumento para a exceção é uma string que indica a causa do erro. Os atributos msg e opt fornecem a mensagem de erro e a opção relacionada.

## Exemplo

Suponha que queiramos passar dois nomes de arquivos pela linha de comando e também queiramos dar uma opção para verificar o uso do script. O uso do script é o seguinte -

```
usage: test.py -i <inputfile> -o <outputfile>
```

Aqui está o seguinte script para test.py -

```
import sys, getopt
def main(argv):
    inputfile = ''
    outputfile = ''
    try:
        opts, args = getopt.getopt(argv,"hi:o:",["ifile=", "ofile="])
    except getopt.GetoptError:
        print ('test.py -i <inputfile> -o <outputfile>')
        sys.exit(2)
    for opt, arg in opts:
        if opt == '-h':
            print ('test.py -i <inputfile> -o <outputfile>')
            sys.exit()
        elif opt in ("-i", "--ifile"):
            inputfile = arg
        elif opt in ("-o", "--ofile"):
            outputfile = arg
    print ('Input file is "', inputfile)
    print ('Output file is "', outputfile)
if __name__ == "__main__":
    main(sys.argv[1:])
```

Agora, execute o script acima da seguinte maneira -

```
$ test.py -h
usage: test.py -i <inputfile> -o <outputfile>
$ test.py -i BMP -o
usage: test.py -i <inputfile> -o <outputfile>
$ test.py -i inputfile -o outputfile
Input file is " inputfile
Output file is " outputfile
```

## O módulo argparse

O módulo **argparse** fornece ferramentas para escrever interfaces de linha de comando muito fáceis de usar. Ele trata de como analisar os argumentos coletados na lista **sys.argv**, gera ajuda automaticamente e emite mensagens de erro quando opções inválidas são fornecidas.

O primeiro passo para projetar a interface da linha de comando é configurar o objeto analisador. Isso é feito pela função `ArgumentParser()` no módulo `argparse`. A função pode receber uma string explicativa como parâmetro de descrição.

Para começar, nosso script será executado na linha de comando sem quaisquer argumentos. Ainda use o método **parse\_args()** do objeto analisador, que não faz nada porque não há argumentos fornecidos.

```
import argparse
parser=argparse.ArgumentParser(description="sample argument parser")
args=parser.parse_args()
```

Quando o script acima é executado -

```
C:\Python311>python parser1.py
C:\Python311>python parser1.py -h
usage: parser1.py [-h]
sample argument parser
options:
  -h, --help show this help message and exit
```

O segundo uso da linha de comando oferece a opção **-help** que produz uma mensagem de ajuda conforme mostrado. O parâmetro **-help** está disponível por padrão.

Agora vamos definir um argumento que é obrigatório para a execução do script e, se não for fornecido, o script deverá gerar um erro. Aqui definimos o argumento 'usuário' pelo método `add_argument()`.

```
import argparse
parser=argparse.ArgumentParser(description="sample argument parser")
parser.add_argument("user")
args=parser.parse_args()
```



```
if args.user=="Admin":
    print ("Hello Admin")
else:
    print ("Hello Guest")
```

A ajuda deste script agora mostra um argumento posicional na forma de 'usuário'. O programa verifica se o valor é 'Admin' ou não e imprime a mensagem correspondente.

```
C:\Python311>python parser2.py --help
usage: parser2.py [-h] user
sample argument parser
positional arguments:
  user
options:
  -h, --help show this help message and exit
```

Use o seguinte comando -

```
C:\Python311>python parser2.py Admin
Hello Admin
```

Mas o uso a seguir exibe a mensagem Hello Guest.

```
C:\Python311>python parser2.py Rajan
Hello Guest
```

## método add\_argument()

Podemos atribuir um valor padrão a um argumento no método add\_argument().

```
import argparse
parser=argparse.ArgumentParser(description="sample argument parser")
parser.add_argument("user", nargs='?',default="Admin")
args=parser.parse_args()
if args.user=="Admin":
    print ("Hello Admin")
else:
    print ("Hello Guest")
```

Aqui, nargs é o número de argumentos de linha de comando que devem ser consumidos. '?'. Um argumento será consumido na linha de comando, se possível, e produzido como um único item. Se nenhum argumento de linha de comando estiver presente, o valor padrão será produzido.

Por padrão, todos os argumentos são tratados como strings. Para mencionar explicitamente o tipo de argumento, use o parâmetro `type` no método `add_argument()`. Todos os tipos de dados Python são valores válidos de tipo.

```
import argparse
parser=argparse.ArgumentParser(description="add numbers")
parser.add_argument("first", type=int)
parser.add_argument("second", type=int)
args=parser.parse_args()
x=args.first
y=args.second
z=x+y
print ('addition of {} and {} = {}'.format(x,y,z))
```

Ele produzirá a seguinte **saída** -

```
C:\Python311>python parser3.py 10 20
addition of 10 and 20 = 30
```

Nos exemplos acima, os argumentos são obrigatórios. Para adicionar um argumento opcional, prefixe seu nome com traço duplo `--`. No caso seguinte, o argumento do sobrenome é opcional porque é prefixado por um traço duplo (`--sobrenome`).

```
import argparse
parser=argparse.ArgumentParser()
parser.add_argument("name")
parser.add_argument("--surname")
args=parser.parse_args()
print ("My name is ", args.name, end=' ')
if args.surname:
    print (args.surname)
```

Um nome de argumento de uma letra prefixado por um único traço atua como uma opção de nome abreviado.

```
C:\Python311>python parser3.py Anup
My name is Anup
C:\Python311>python parser3.py Anup --surname Gupta
My name is Anup Gupta
```

Se for desejado que um argumento tenha valor apenas de uma lista definida, ele é definido como parâmetro de escolhas.

```
import argparse
parser=argparse.ArgumentParser()
```



```
parser.add_argument("sub", choices=['Physics', 'Maths', 'Biology'])
args=parser.parse_args()
print ("My subject is ", args.sub)
```

Observe que se o valor do parâmetro não estiver na lista, um erro de escolha inválida será exibido.

```
C:\Python311>python parser3.py Physics
```

```
My subject is Physics
```

```
C:\Python311>python parser3.py History
```

```
usage: parser3.py [-h] {Physics,Maths,Biology}
```

```
parser3.py: error: argument sub: invalid choice: 'History' (choose from  
'Physics', 'Maths', 'Biology')
```