How to create a command line program in Python with Click

Click is a Python package to write command line interfaces with as little code as possible. This article will show you how to use Click to create the command line program.

Click is a Python package to write command line interfaces with as little code as possible. This article will show you how to use Click to create the command line program.

How to create a command line program in Python with Click

1. Write a command line program without Click
2. Get familiar with Click in Python
3. Write the first Click program
4. Add another command
5. Other Click options

Write a command line program without Click

You can write command line programs without clicking but requires more skills and code. You need to parse the command, perform authentication, develop logic to handle various parameters and build a custom help menu. If you want to add a new option, you need to modify the help function.

It is useful to be able to write your own code, this is a great way to learn Python programming language, but Click allows you to follow the 'Don't Repeat Yourself' principle (Don't repeat anything). same - DRY). If you do not use Click, you will have to write the code yourself and maintain it whenever there is a change.

This is a simple command line interface that is code without clicking:

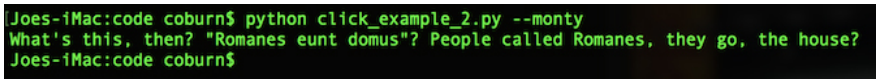
```
import sys
import random

def do_work ():
"" "Function to handle command line usage" ""
args = sys.argv
args = args [1:] # First element of args is the file name

if len (args) == 0:
in ra ('B?n không ??a ra m?t câu l?nh trong!')
else:
```

```
cho m?t trong args:
if a == '--help':
print ('Basic command line program')
print ('Options:')
print ('--help -> show this basic help menu.')
print ('--monty -> show a Monty Python quote.')
print ('--veg -> show a random vegetable')
elif a == '--monty':
print ('What's this, then? "Romanes eunt domus?" People called Romanes, they g
elif a == '--veg':
print (random.choice (['Carrot', 'Potato', 'Turnip']))
else:
print ('Unrecognised argument.')

if __name__ == '__main__':
do_work ()
```

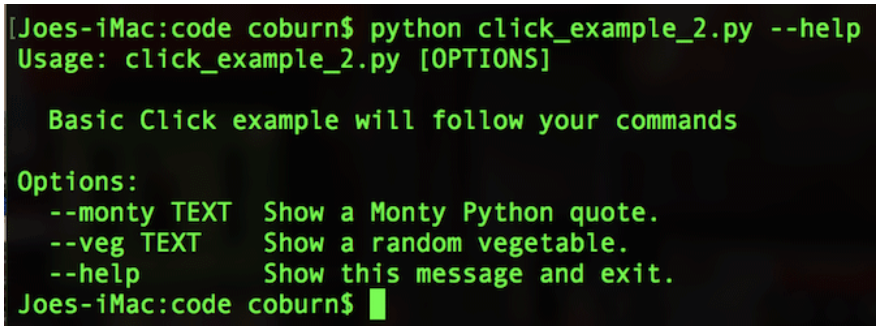


These 27 Python lines work well but are fragile. When making any changes on the program, you also need to change the other support code. If you change a parameter name, you need to update the help information and often get confused.

This is the same logic but used with Click:

```
import click
import random
@ click.command ()
@ click.option ('- monty', default = False, help = 'Show a Monty Python quote.
@ click.option ('- veg', default = False, help = 'Show a random vegetable.')
def do_work (monty, veg):
"" "Basic Click example will follow your commands" ""
if monty:
print ('What's this, then? "Romanes eunt domus?" People called Romanes, they g
if veg:
print (random.choice (['Carrot', 'Potato', 'Turnip']))
if __name__ == '__main__':
do_work ()
```

You can see that Click performs the same logic but only with 16 lines of code instead of the 27 lines of code above. And you don't need parametric analysis and it will create a help screen.



This is just a basic comparison example so you can see that using programs like Click will save you a lot of time and effort. Although the command line interface to the end user is the same, the code that makes up the program is simpler, saving you a lot of time writing code.

Get familiar with Click in Python

Before using Click, you can configure a virtual environment. This will prevent Python packages from conflicting with the Python system or other projects you are working on. You can try Python in your browser if you want to learn more about Python and Click.

Finally, make sure you're running Python version 3. Although Click can be used with Python version 2, the examples here are in Python 3.

Click Settings from the command line using PIP.

```
pip install click
```

Write the first Click program

In the text editor, start by entering Click:

```
import click
```

After entering Click, create a **method** and **main** entry point.

```
import click
import random

def veg ():
"" "Basic method will return a random vegetable" ""
print (random.choice (['Carrot', 'Potato', 'Turnip', 'Parsnip']))

if __name__ == '__main__':
veg ()
```

This simple script will produce a random vegetable. Your code may be different but this simple example is the perfect way to connect to Click.

Save it to **click_example.py** and then run it in the command line (after navigating to its location):

```
> python click_example.py
```

You will see the vegetable name randomly, now improve this code by adding Click. Change your code to contain decorators and **for** in Python loops:

```
@ click.command ()
@ click.option ('- total', default = 3, help = 'Number of vegetables to output
def veg (total):
"" "Basic method will return a random vegetable" ""
cho s? trong ph?m vi (total):
print (random.choice (['Carrot', 'Potato', 'Turnip', 'Parsnip']))

if __name__ == '__main__':
veg ()
```

When running, you will see a random vegetable displayed three times. Let's analyze the above code.

Decorator **@ click.command ()** configures Click to work with Python functions right after decorator. In this case, this is the **veg ()** function. You need it for the methods used with Click.

Decorator **@ click.option** configuration Click to accept the parameter from the command line, move to your method. There are three parameters used here:

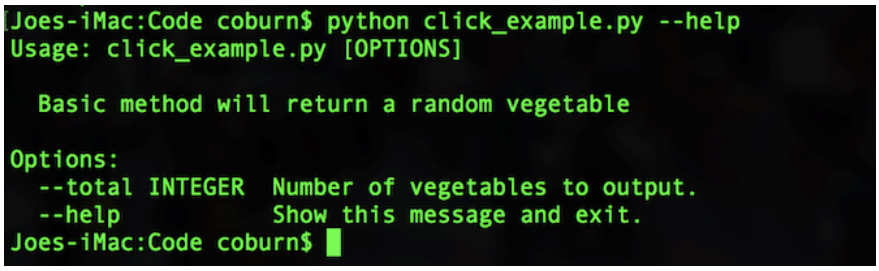
1. **–Total** : This is the command line name for the **total** parameter.
2. **default** : If you don't specify the total parameter when using the script, Click will use the default value.
3. **help** : This is a short sentence that explains how to use the program.

See the actual example using Click. From the command line, run the script but pass the total parameter as follows:

```
python click_example.py --total 10
```

When the **–total** setting is **10** from the command line, the script will print 10 random vegetables. If you use the **-help** flag, you will see a help page with options you can use:

```
python click_example.py --help
```



Add another command

You can use multiple Click decorators on the same function. Add another Click option to the **veg** function as follows:

```
@ click.option ('- gravy', default = False, help = 'Append "with gravy" to the
```

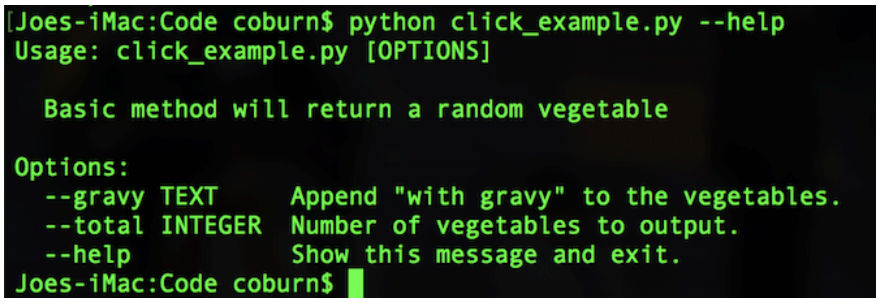
However, do not forget to pass it into the method:

```
def veg (total, gravy):
```

Now when you run the file, you can transfer it in the flag **gravy** :

```
python click_example.py --gravy y
```

The help screen also changes:



This is the whole code (with some small code changes to look more neat):

```
import click
import random

@ click.command ()
@ click.option ('- gravy', default = False, help = 'Append "with gravy" to the
@ click.option ('- total', default = 3, help = 'Number of vegetables to output
def veg (total, gravy):
"" "Basic method will return a random vegetable" ""
cho s? trong ph?m vi (total):
choice = random.choice (['Carrot', 'Potato', 'Turnip', 'Parsnip'])

if gravy:
print (f '{choice} with gravy')
else:
print (choice)

if __name__ == '__main__':
veg ()
```

Other Click options

Once you know the basics of Clicking, you can start to look at the more complex Click options. In this example, you will learn how to pass some values ??into a parameter to Click convert to a tuple data type.

Create a new file named **click_example_2.py** . This is the code you need:

```
import click
import random
```

```
@ click.command ()
def add ():
"" Basic method will add two numbers together. "" "
pass

if __name__ == '__main__':
add ()
```

Then add **@ click.option** called **numbers** :

```
@ click.option ('- numbers'>, nargs = 2, type = int, help = 'Add two numbers t
```

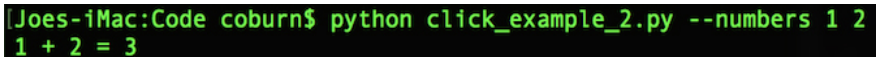
The only new code here is **nargs = 2** , and the **type = int option** for Click to accept two values ??for the **numbers** option and both must be integers. You can change any number or value type (valid) you like.

Finally, change the **add** method to accept the parameter parameter and make some changes to them:

```
def add (numbers):
"" Basic method will add two numbers together. "" "
result = numbers [0] + numbers [1]
print (f '{numbers [0]} + {numbers [1]} = {result}')
```

Each value you pass can be accessed via object **numbers** . Here's how to use it in the command line:

```
python click_example_2.py --numbers 1 2
```



I wish you all success!

See more:

1. How to install Python on Windows, macOS, Linux
2. Learn the first Python program
3. More than 100 Python exercises have solutions (sample code)