Control Flow and Function

Olarik Surinta, PhD.

Outline

- if statement
- for statements
- while statements
- Simple defining functions
- argparse

Control Flow: if Statements

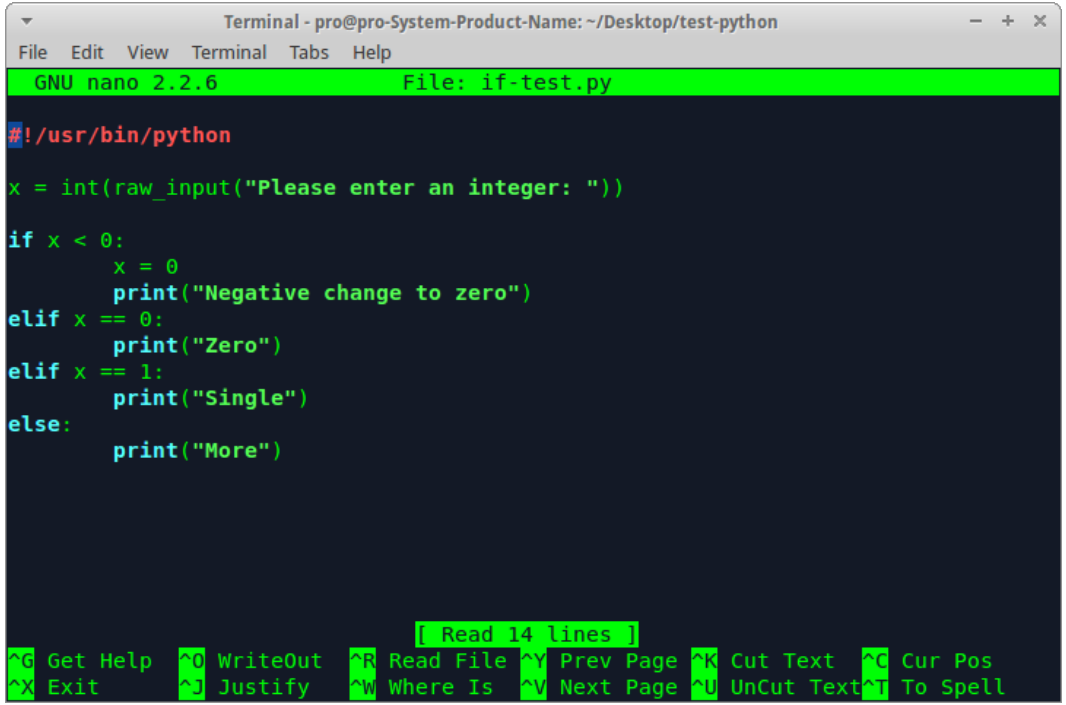
```
#!/usr/bin/python

x = int(raw_input("Please enter an integer: "))

if x < 0:
        x = 0
        print("Negative change to zero")
elif x == 0:
        print("Zero")
elif x == 1:
        print("Single")
else:
        print("More")
```

$ sudo nano if-test.py

Control Flow: if Statements

$ chmod +x if-test.py

$ ./if-test.py



if statements

```
weight = float(input("How many pounds does your suitcase weigh? "))
if weight > 50:
    print("There is a $25 charge for luggage that heavy.")
print("Thank you for your business.")
```

if statements

```
temperature = float(input('What is the temperature? '))
if temperature > 70:
    print('Wear shorts.')
else:
    print('Wear long pants.')
print('Get some exercise outside.')
```

Conditional Expressions

- There should not be space between two-symbol Python substitutes.

Meaning	Math Symbol	Python Symbols
Less than	<	<
Greater than	>	>
Less than or equal	≤	<=
Greater than or equal	≥	>=
Equals	=	==
Not equal	≠	!=

if statements

```
score = float(input('Enter \
examination score: '))
if score >= 90:
    letter = 'A'
elif score >= 80:
    letter = 'B'
elif score >= 70:
    letter = 'C'
elif score >= 60:
    letter = 'D'
else:
    letter = 'F'
print("Your grade is %s"% \
(letter))
```

Simple Defining Functions

- **A First Function Definition**

```
def function_name():
```

```
def letterGrade(score):
    if score >= 90:
        letter = 'A'
    elif score >= 80:
        letter = 'B'
    elif score >= 70:
        letter = 'C'
    elif score >= 60:
        letter = 'D'
    else:
        letter = 'F'
    return letter
```

Terminal - pro@pro-System-Product-Name: ~/Desktop/test-python

File Edit View Terminal Tabs Help

GNU nano 2.2.6 File: if-grade-def.py Modified

```
#!/usr/bin/python

def letterGrade(score):
    if score >= 90:
        letter = 'A'
    elif score >= 80:
        letter = 'B'
    elif score >= 70:
        letter = 'C'
    elif score >= 60:
        letter = 'D'
    else:
        letter = 'F'
    return letter
```

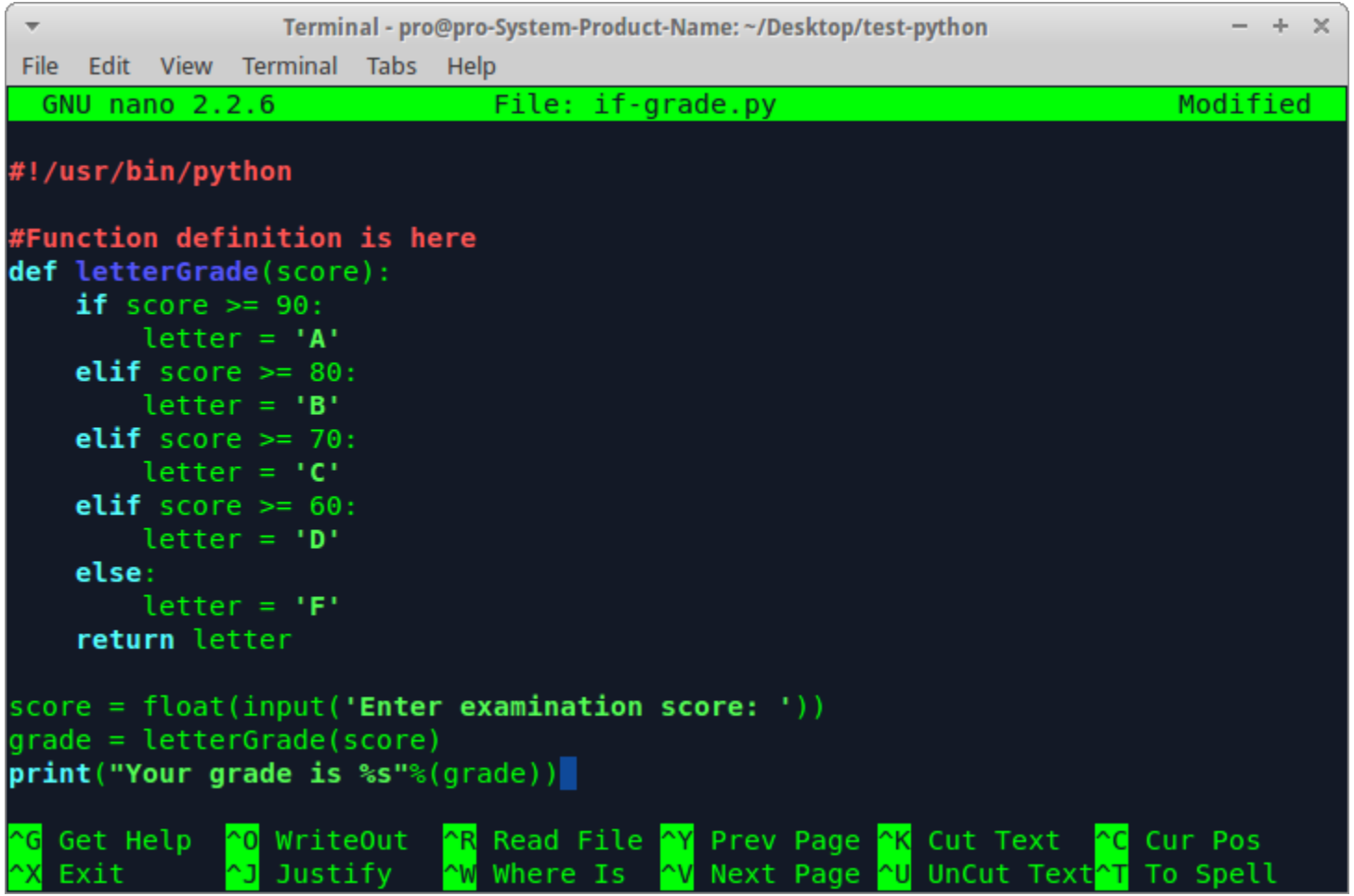
[Read 19 lines]

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

Simple Defining Functions

- **Note:** The statements in the function *definition* are not executed as Python first passes over the lines.
- Calling a Function

```
score = float(input('Enter examination score: '))
grade = letterGrade(score)
print("Your grade is %s"%(grade))
```


Terminal - pro@pro-System-Product-Name: ~/Desktop/test-python
File Edit View Terminal Tabs Help
GNU nano 2.2.6 File: if-grade.py Modified
#!/usr/bin/python
#Function definition is here
def letterGrade(score):
if score >= 90:
letter = 'A'
elif score >= 80:
letter = 'B'
elif score >= 70:
letter = 'C'
elif score >= 60:
letter = 'D'
else:
letter = 'F'
return letter
score = float(input('Enter examination score: '))
grade = letterGrade(score)
print("Your grade is %s"%(grade))
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

Multiple Function Definitions

```
''' Function definitions and invocation. '''
def helloWorld():
    print("Hello World!")
def helloTeacher():
    print("Good morning teacher!!")

def main():
    helloWorld()
    helloTeacher()

main()
```

Terminal - pro@pro-System-Product-Name: ~/Desktop/test-python

File Edit View Terminal Tabs Help

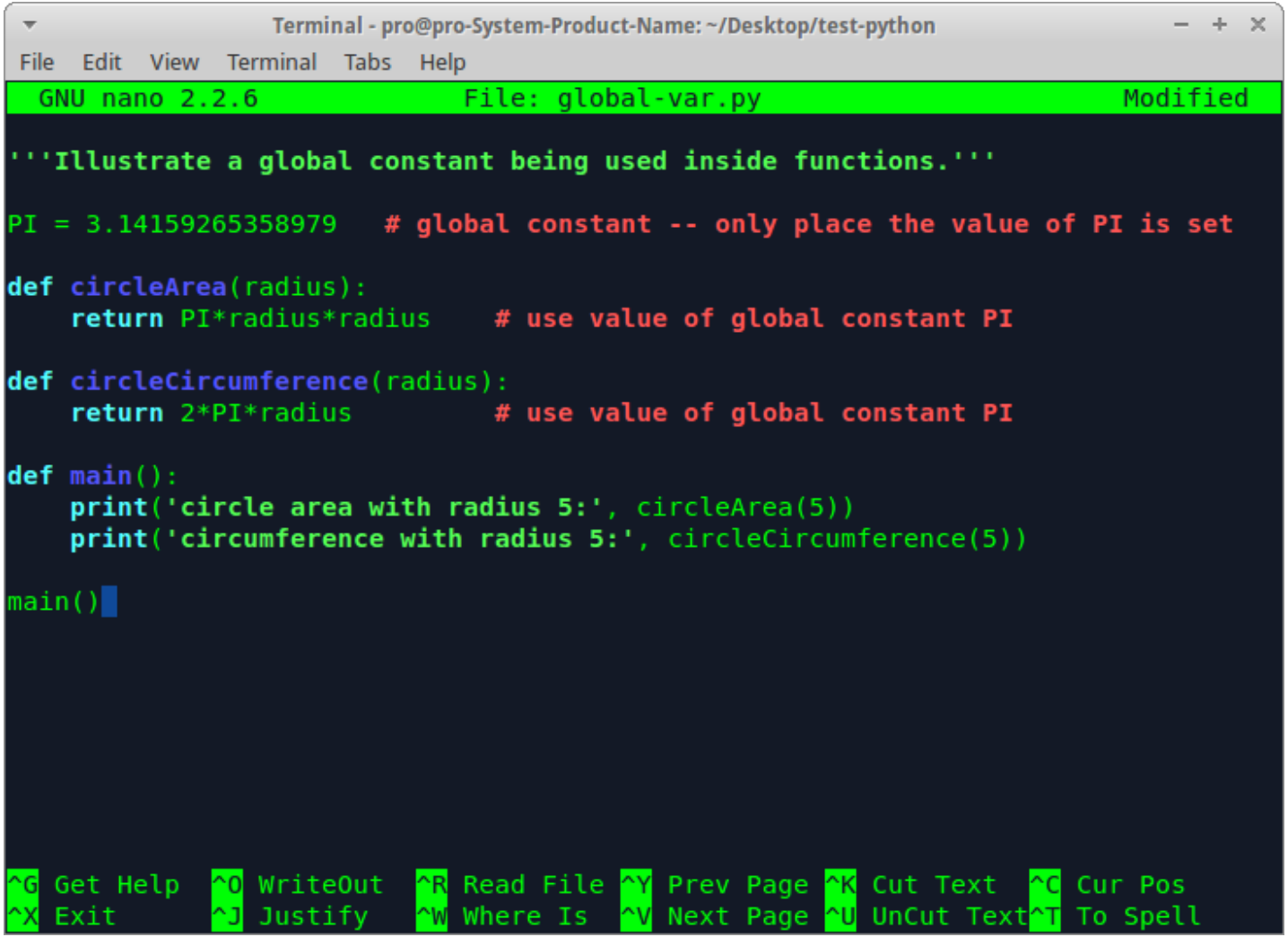
GNU nano 2.2.6 File: if-grade-def.py Modified

```
#!/usr/bin/python

def letterGrade(score):
    if score >= 90:
        letter = 'A'
    elif score >= 80:
        letter = 'B'
    elif score >= 70:
        letter = 'C'
    elif score >= 60:
        letter = 'D'
    else:
        letter = 'F'
    return letter


def main():
    score = float(input('Enter examination score: '))
    grade = letterGrade(score)
    print("Your grade is %s"%(grade))

main()
```

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

Global Constants

- If you define global variables (variables defined outside of any function definition), they are visible inside all of your functions. They have global scope.


Terminal - pro@pro-System-Product-Name: ~/Desktop/test-python
File Edit View Terminal Tabs Help
GNU nano 2.2.6 File: global-var.py Modified
'''Illustrate a global constant being used inside functions.'''
PI = 3.14159265358979 # global constant -- only place the value of PI is set
def circleArea(radius):
return PI*radius*radius # use value of global constant PI
def circleCircumference(radius):
return 2*PI*radius # use value of global constant PI
def main():
print('circle area with radius 5:', circleArea(5))
print('circumference with radius 5:', circleCircumference(5))
main()
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

for statements

- The for statement in Python differs a bit from what you may be used in C.

```
>>> words = ['cat', 'dog', 'window']
>>> for w in words:
…       print w, len(w)
…
cat 3
dog 3
window 6
```

Simple Defining Functions

- **A First Function Definition**

```
def function_name():
```

```
def letterGrade(score):
    if score >= 90:
        letter = 'A'
    elif score >= 80:
        letter = 'B'
    elif score >= 70:
        letter = 'C'
    elif score >= 60:
        letter = 'D'
    else:
        letter = 'F'
    return letter
```

Terminal - pro@pro-System-Product-Name: ~/Desktop/test-python

File Edit View Terminal Tabs Help

GNU nano 2.2.6 File: if-grade-def.py Modified

```
#!/usr/bin/python

def letterGrade(score):
    if score >= 90:
        letter = 'A'
    elif score >= 80:
        letter = 'B'
    elif score >= 70:
        letter = 'C'
    elif score >= 60:
        letter = 'D'
    else:
        letter = 'F'
    return letter
```

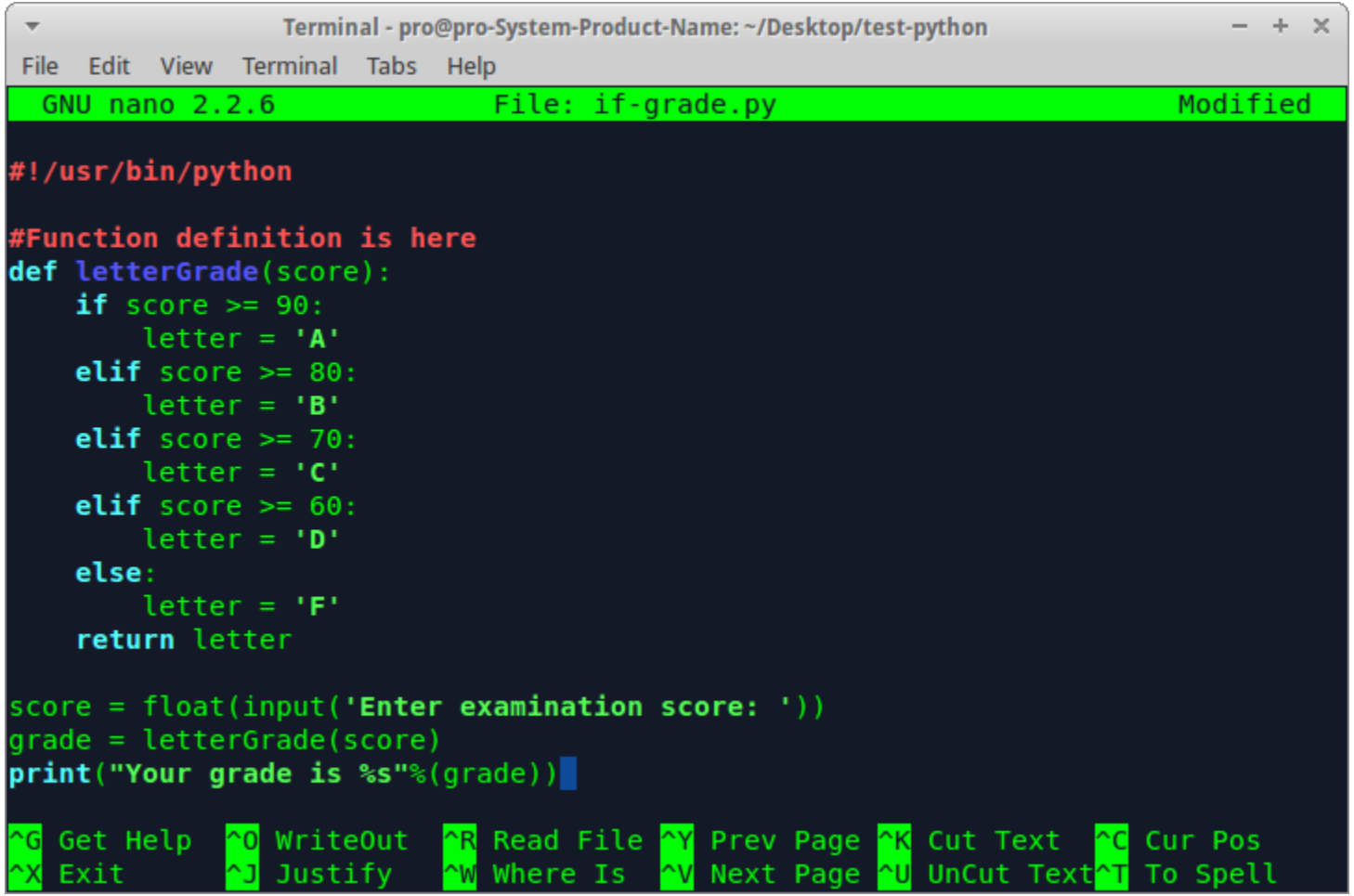
[Read 19 lines]

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

Simple Defining Functions

- **Note:** The statements in the function *definition* are not executed as Python first passes over the lines.
- Calling a Function

```
score = float(input('Enter examination score: '))

grade = letterGrade(score)

print("Your grade is %s"%(grade))
```


Terminal - pro@pro-System-Product-Name: ~/Desktop/test-python
File Edit View Terminal Tabs Help
GNU nano 2.2.6 File: if-grade.py Modified
#!/usr/bin/python
#Function definition is here
def letterGrade(score):
if score >= 90:
letter = 'A'
elif score >= 80:
letter = 'B'
elif score >= 70:
letter = 'C'
elif score >= 60:
letter = 'D'
else:
letter = 'F'
return letter
score = float(input('Enter examination score: '))
grade = letterGrade(score)
print("Your grade is %s"%(grade))
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

Multiple Function Definitions

```
''' Function definitions and invocation. '''
def helloWorld():
    print("Hello World!")
def helloTeacher():
    print("Good morning teacher!!")

def main():
    helloWorld()
    helloTeacher()

main()
```

Terminal - pro@pro-System-Product-Name: ~/Desktop/test-python

File Edit View Terminal Tabs Help

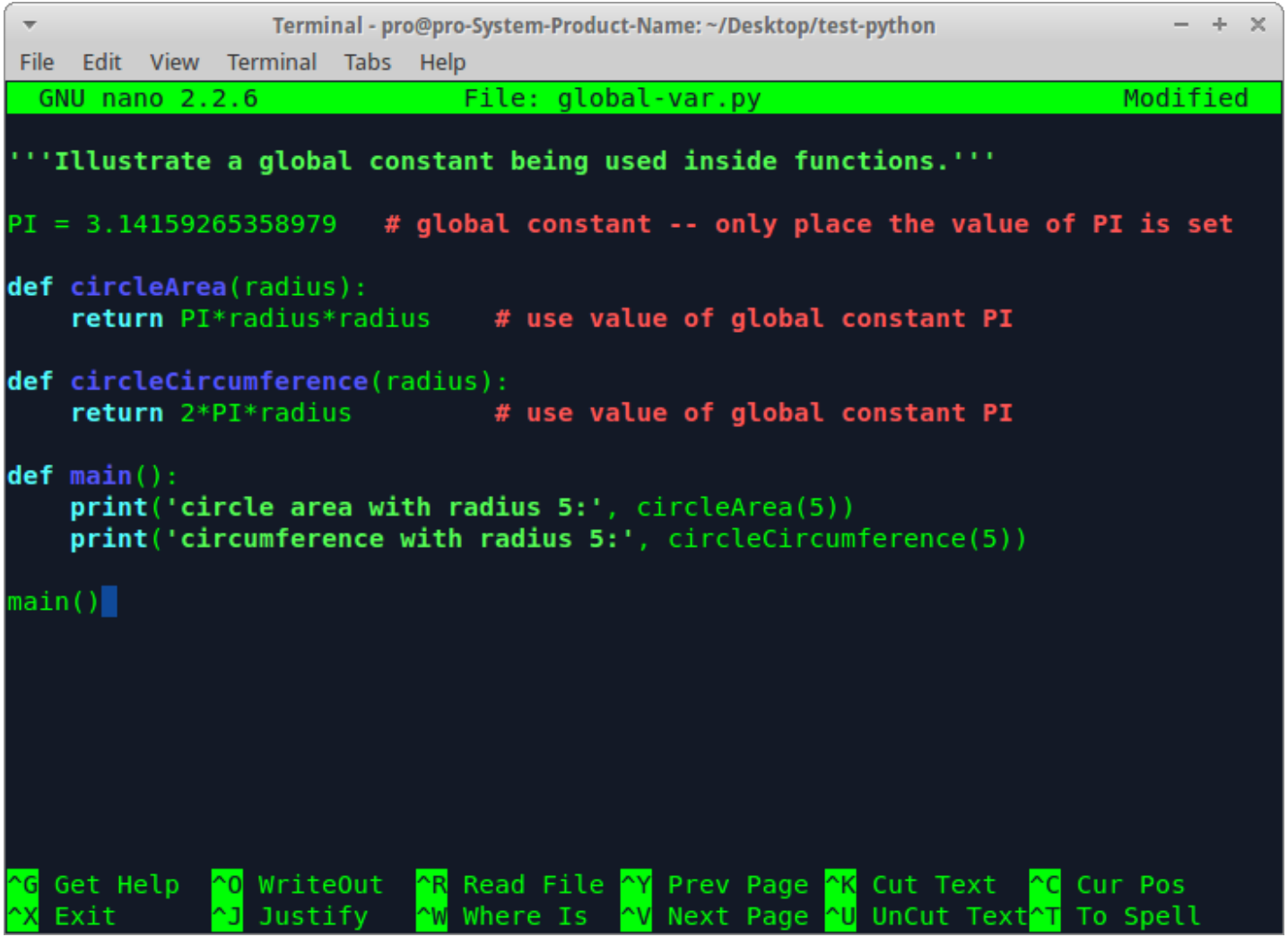
GNU nano 2.2.6 File: if-grade-def.py Modified

```
#!/usr/bin/python

def letterGrade(score):
    if score >= 90:
        letter = 'A'
    elif score >= 80:
        letter = 'B'
    elif score >= 70:
        letter = 'C'
    elif score >= 60:
        letter = 'D'
    else:
        letter = 'F'
    return letter


def main():
    score = float(input('Enter examination score: '))
    grade = letterGrade(score)
    print("Your grade is %s"%(grade))

main()
```

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

Global Constants

- If you define global variables (variables defined outside of any function definition), they are visible inside all of your functions. They have global scope.


Terminal - pro@pro-System-Product-Name: ~/Desktop/test-python
File Edit View Terminal Tabs Help
GNU nano 2.2.6 File: global-var.py Modified
'''Illustrate a global constant being used inside functions.'''
PI = 3.14159265358979 # global constant -- only place the value of PI is set
def circleArea(radius):
return PI*radius*radius # use value of global constant PI
def circleCircumference(radius):
return 2*PI*radius # use value of global constant PI
def main():
print('circle area with radius 5:', circleArea(5))
print('circumference with radius 5:', circleCircumference(5))
main()
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

for statements

```
>>> for w in words[:]:
...         if len(w) > 5:
...             words.insert(0,w)
...
>>> words
['window', 'cat', 'dog', 'window']
```

for statements

```
for letter in 'Python':
    print 'Current Letter : ', letter
```

```
Current Letter : P
Current Letter : y
Current Letter : t
...
```

For statements

```
n = 100


sum = 0
for counter in range(1,n+1):
    sum = sum + counter


print("Sum of 1 until %d: %d" % (n,sum))
```

Sum of 1 until 100: 5050

range() Function

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(5, 10)
[5, 6, 7, 8, 9]
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> range(-10, -100, -30)
[-10, -40, -70]
```

range(), len()

```
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
>>> for i in range(len(a)):
...     print i, a[i]
...
0 Mary
1 had
2 a
3 little
4 lamb
```

Break statements

- The **break** statement breaks out of the smallest enclosing **for** or **while** loop.

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print n, 'equals', x, '*', n/x
...             break
...     else:
...         # loop fell through without finding a factor
...         print n, 'is a prime number'
...
```

```
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

While statements

```
count = 0
While (count <9):
    print("The count is: %d"%(count))
    count = count + 1

Print("Good bye!")
```

The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!

While statements

```
n = raw_input("Please enter 'hello':")

while n.strip() != 'hello':
    n=raw_input("Please enter 'hello':")
```

>>> n = 'Olarik Surinta'
>>> n.strip()
'Olarik Surinta'

Please enter 'hello':olarik
Please enter 'hello':surinta
Please enter 'hello':hello

While statements

```
while True:
    n = raw_input("Please enter 'hello':")
    if n.strip() == 'hello':
        break
```

While statements

```
password = ""
while password != "secret":
    password = raw_input("Please enter the password: ")
    if password == "secret":
        print("Thank you. You have entered the correct password")
    else:
        print("Sorry the value entered in incorrect - try again")
```

Check if raw input is interger

```
y = x.isdigit()
if (y == False):
    print('not digit')
else:
    print('digit')
```

Enter a number or a word: 4
digit

Enter a number or a word: a
Not digit

Check if raw input is interger

```
try:
    val = int(raw_input("number: "))
except ValueError:
    print("error")
```

While Statements

```
invalid = True
while invalid:
    number = int(input("Please enter a number in the range 10 to 20: "))
    if number >= 10 and number <= 20:
        invalid = False
    else:
        print("Sorry number must be between 10 and 20")
        print("Please try again")
print("You entered {0}".format(number))
print("This is a valid number")
```

Interactive while statements

```
lines = list()
n = int(input('How many lines do you want to enter? '))
for i in range(n):
    line = input('Next line: ')
    lines.append(line)

print('Your lines were:')  # check now
for line in lines:
    print(line)
```

Interactive while statements

```
lines = list()
testAnswer = raw_input('Press y if you want to enter more lines: ')
while testAnswer == 'y':
    line = input('Next line: ')
    lines.append(line)
    testAnswer = input('Press y if you want to enter more lines: ')

print('Your lines were:')
for line in lines:
    print(line)
```

Switch-case statements

- Unlike every other programming language, Python does not have a switch or case statement.

Switch-case statements

```
def zero():
    print "You typed zero.\n"

def sqr():
    print "n is a perfect square\n"

def even():
    print "n is an even number\n"

def prime():
    print "n is a prime number\n"
```

Switch-case statements

```
options = {0 : zero,
           1 : sqr,
           4 : sqr,
           9 : sqr,
           2 : even,
           3 : prime,
           5 : prime,
           7 : prime,
}

options[1]()
```

Simple Defining Functions

- **A First Function Definition**

```
def function_name():
```

```
def letterGrade(score):
    if score >= 90:
        letter = 'A'
    elif score >= 80:
        letter = 'B'
    elif score >= 70:
        letter = 'C'
    elif score >= 60:
        letter = 'D'
    else:
        letter = 'F'
    return letter
```

Simple Defining Functions

- **A First Function Definition**

```
def function_name():
```

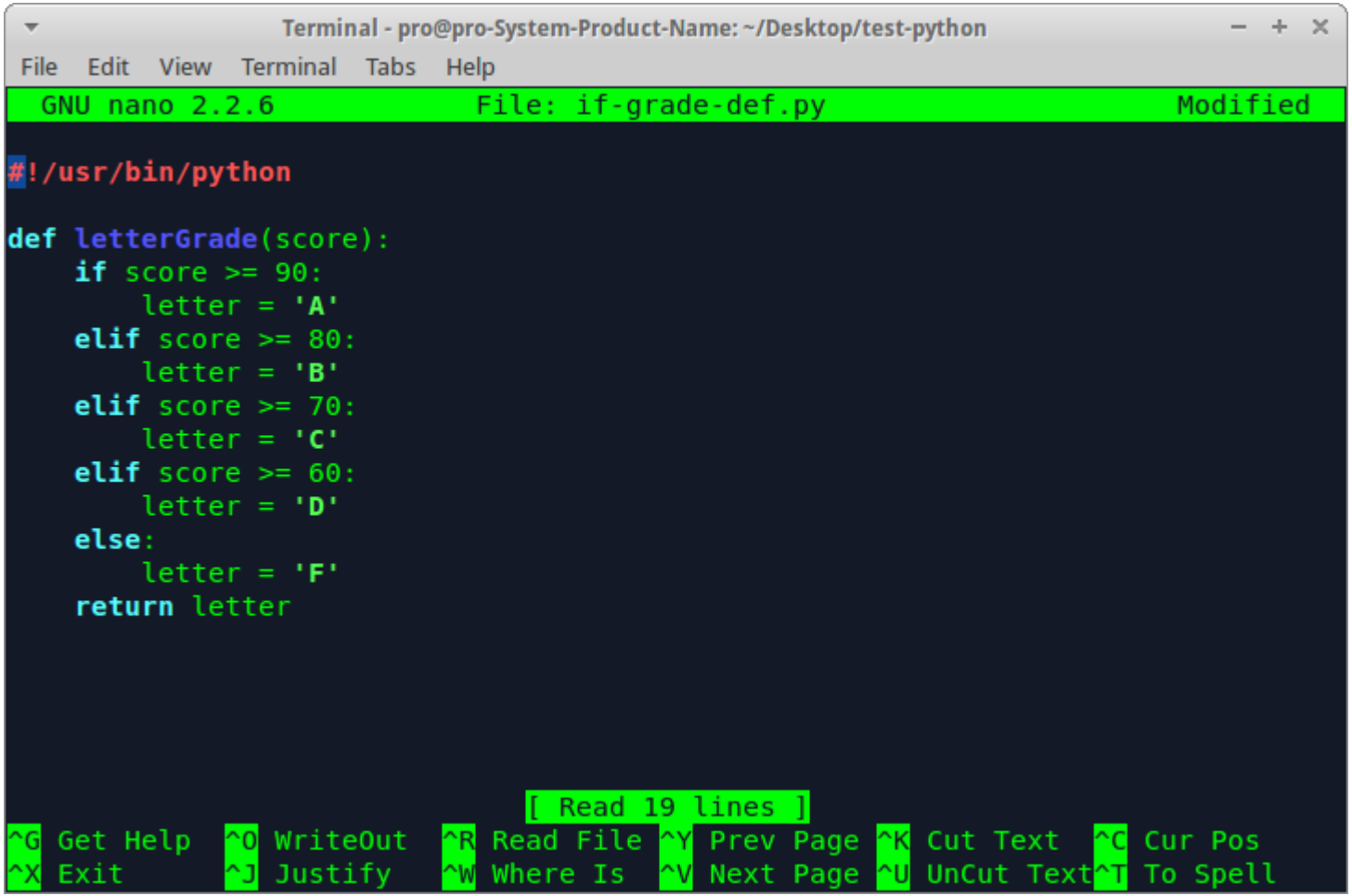
```
def letterGrade(score):
    if score >= 90:
        letter = 'A'
    elif score >= 80:
        letter = 'B'
    elif score >= 70:
        letter = 'C'
    elif score >= 60:
        letter = 'D'
    else:
        letter = 'F'
    return letter
```

Simple Defining Functions

- **A First Function Definition**

```
def function_name():
```

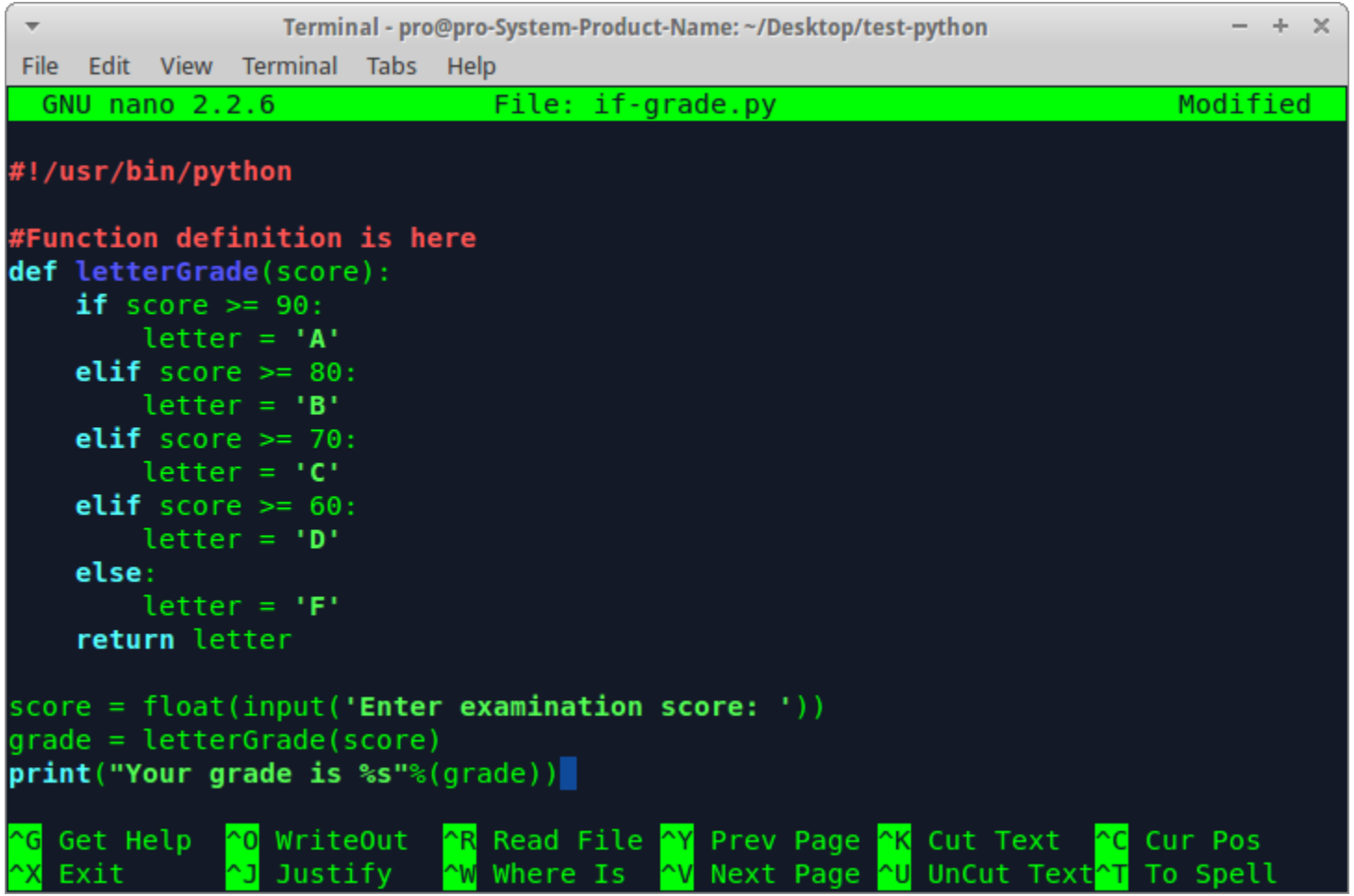
```
def letterGrade(score):
    if score >= 90:
        letter = 'A'
    elif score >= 80:
        letter = 'B'
    elif score >= 70:
        letter = 'C'
    elif score >= 60:
        letter = 'D'
    else:
        letter = 'F'
    return letter
```


Terminal - pro@pro-System-Product-Name: ~/Desktop/test-python
File Edit View Terminal Tabs Help
GNU nano 2.2.6 File: if-grade-def.py Modified
#!/usr/bin/python
def letterGrade(score):
if score >= 90:
letter = 'A'
elif score >= 80:
letter = 'B'
elif score >= 70:
letter = 'C'
elif score >= 60:
letter = 'D'
else:
letter = 'F'
return letter
[Read 19 lines]
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

Simple Defining Functions

- **Note:** The statements in the function *definition* are not executed as Python first passes over the lines.
- Calling a Function

```
score = float(input('Enter examination score: '))
grade = letterGrade(score)
print("Your grade is %s"%(grade))
```


Terminal - pro@pro-System-Product-Name: ~/Desktop/test-python
File Edit View Terminal Tabs Help
GNU nano 2.2.6 File: if-grade.py Modified
#!/usr/bin/python
#Function definition is here
def letterGrade(score):
if score >= 90:
letter = 'A'
elif score >= 80:
letter = 'B'
elif score >= 70:
letter = 'C'
elif score >= 60:
letter = 'D'
else:
letter = 'F'
return letter
score = float(input('Enter examination score: '))
grade = letterGrade(score)
print("Your grade is %s"%(grade))
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

Multiple Function Definitions

```
''' Function definitions and invocation. '''
def helloWorld():
    print("Hello World!")
def helloTeacher():
    print("Good morning teacher!!")

def main():
    helloWorld()
    helloTeacher()

main()
```

Terminal - pro@pro-System-Product-Name: ~/Desktop/test-python

File Edit View Terminal Tabs Help

GNU nano 2.2.6 File: if-grade-def.py Modified

```
#!/usr/bin/python

def letterGrade(score):
    if score >= 90:
        letter = 'A'
    elif score >= 80:
        letter = 'B'
    elif score >= 70:
        letter = 'C'
    elif score >= 60:
        letter = 'D'
    else:
        letter = 'F'
    return letter


def main():
    score = float(input('Enter examination score: '))
    grade = letterGrade(score)
    print("Your grade is %s"%(grade))

main()
```

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

Global Constants

- If you define global variables (variables defined outside of any function definition), they are visible inside all of your functions. They have global scope.

Terminal - pro@pro-System-Product-Name: ~/Desktop/test-python

File Edit View Terminal Tabs Help

GNU nano 2.2.6 File: global-var.py Modified

```
'''Illustrate a global constant being used inside functions.'''

PI = 3.14159265358979   # global constant -- only place the value of PI is set

def circleArea(radius):
    return PI*radius*radius    # use value of global constant PI

def circleCircumference(radius):
    return 2*PI*radius         # use value of global constant PI

def main():
    print('circle area with radius 5:', circleArea(5))
    print('circumference with radius 5:', circleCircumference(5))

main()
```

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

Documentation

```
>>> def usage():
...     """Do nothing, but document it.
...
...     No, really, it doesn't do anything.
...     """
...
>>> print usage.__doc__
Do nothing, but document it.

    No, really, it doesn't do anything.
```

argparse – command line option and argument parsing

- Available in: python 2.7 and later
- Once all of the arguments are defined, you can parse the command line by passing a sequence of argument strings to **parse_args()**.

argparse: passing a command line

```
import argparse

parser = argparse.ArgumentParser(description='Short sample app')

parser.add_argument('-a', action="store_true", default=False)
parser.add_argument('-b', action="store", dest="b")
parser.add_argument('-c', action="store", dest="c", type=int)

print parser.parse_args(['-a', '-bval', '-c', '3'])
```

argparse: passing a command line

```
$ python argparse.py
Namespace(a=True, b='va', c=3)
```

argparse: sample

```
import argparse

# construct the argument parse and parse the arguments
parser = argparse.ArgumentParser(description='Template Matching')
parser.add_argument('-i', '--image', dest='image', required=True, help='Select an input image')
parser.add_argument('-t', '--template', dest='template', required=True, help='Select a template \
        image')
parser.add_argument('--version', action='version', version='%(prog)s version 0.1')
args = vars(parser.parse_args())

print(args['image'])
print(args['template'])
```

argparse: sample

```
$ python argparse1.py -h
usage: par.py [-h] -i IMAGE -t TEMPLATE [--version]

Template Matching

optional arguments:
  -h, --help            show this help message and exit
  -i IMAGE, --image IMAGE
                        Select an input image
  -t TEMPLATE, --template TEMPLATE
                        Select a template image
  --version             show program's version number and exit
```

argparse: sample

$ python argparse1.py -i image.png -t template.png

image.png

template.png

Reading and Writing Files in Python

- Mode
  - ‘r’ - Read mode which is used when the file is only being read
  - ‘w’ - Write mode which is used to edit and write new information to the file
  - ‘a’ - Appending mode, which is used to add new data to the end of the file. The new line information is automatically amended to the end
  - ‘r+’ - Special read and write mode, which is used to handle both actions when working with a file

Create a Text File

```
file = open("testfile.txt","w")

file.write("Hello World")
file.write("This is our new text file ")

file.close()
```

Create a Text File

$ cat testfile.txt

Hello World

This is our new text file

#create a simple script (**run.py**)

```
#!/usr/bin/env python


file = open('show.txt','w')
for x in xrange(1,5000000):
    for y in xrange(1,13):
        print '%d*%d = %d'%(x,y,x*y)
        file.write('%d*%d = %d\n'%(x,y,x*y))


file.close()
```

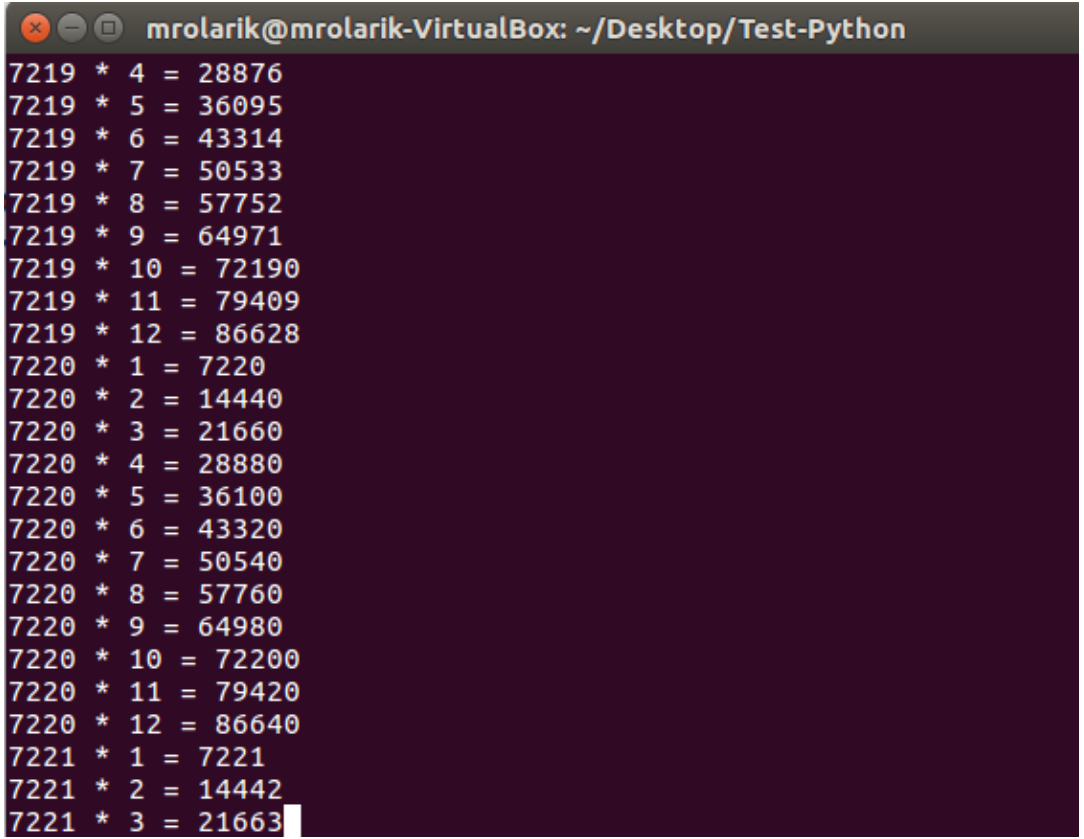
#testing “run.py” program

$ python run.py

the process will

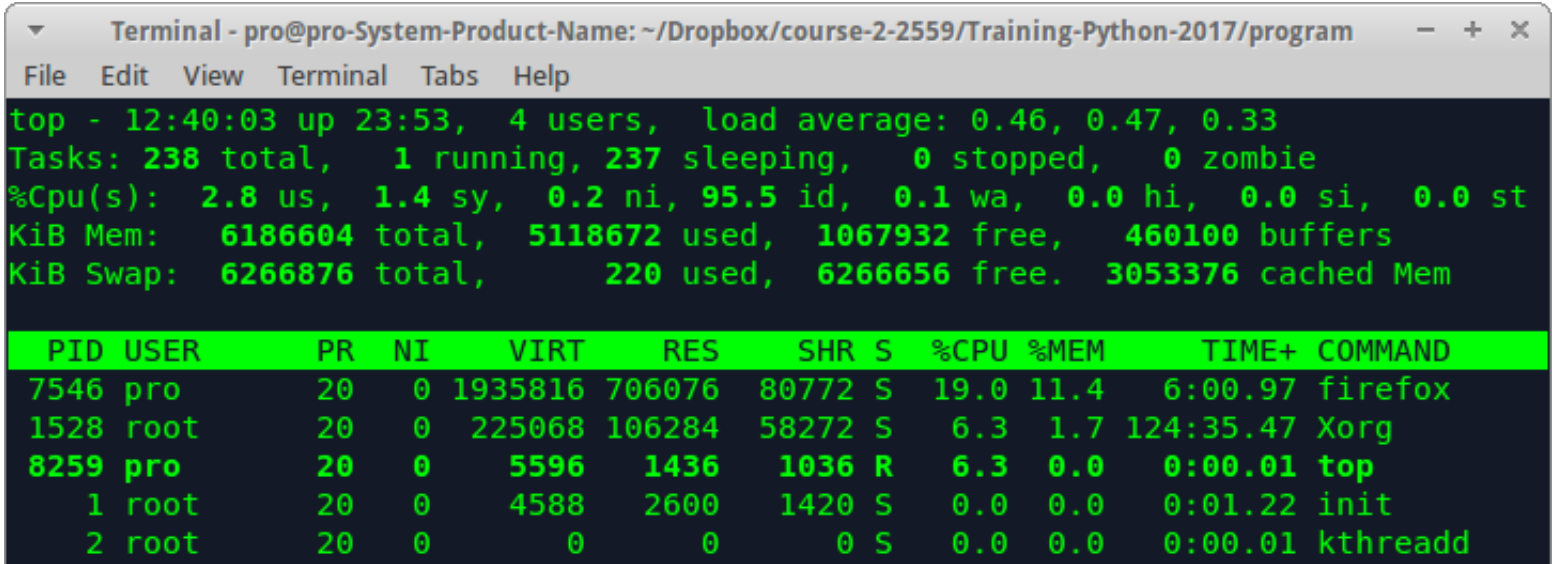
end when you

Close the terminal



top command

- Display and update sorted information about processes. **q = quit**



View a specific process

$ top | grep firefox

```
 7546 pro       20   0 1938112 669700  80848 S  12.9 10.8
6:42.16 firefox
```

$ ps -x | grep firefox

```
 7546 ?         Sl     6:44 /usr/lib/firefox/firefox
 8316 pts/21    S+     0:00 grep --color=auto firefox
```

View a specific process

$ ps -x | grep run.py

8339 pts/20 S+ 0:01 python run.py

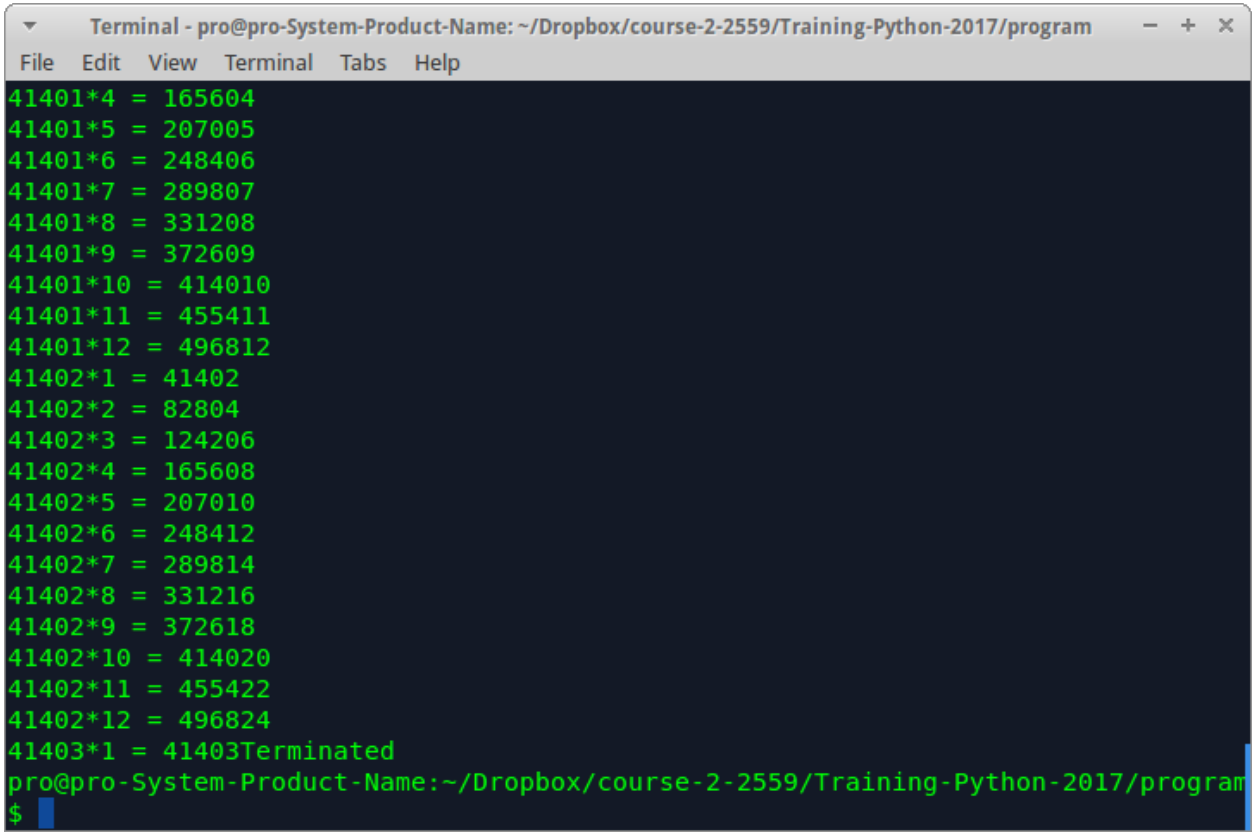
8343 pts/21 S+ 0:00 grep --color=auto run.py

$ pidof python

8339 7006 2508 2497

kill a process

$ kill 8339



Running a program in the background

- **Nohup** is a POSIX command to ignore the HUP (hangup) signal.
- The HUP signal is , by convention, the way a terminal dependent processes of logout.
- Output that would normally go to the terminal goes to a file called “**nohup.out”** if it has not already been redirected.

Usage

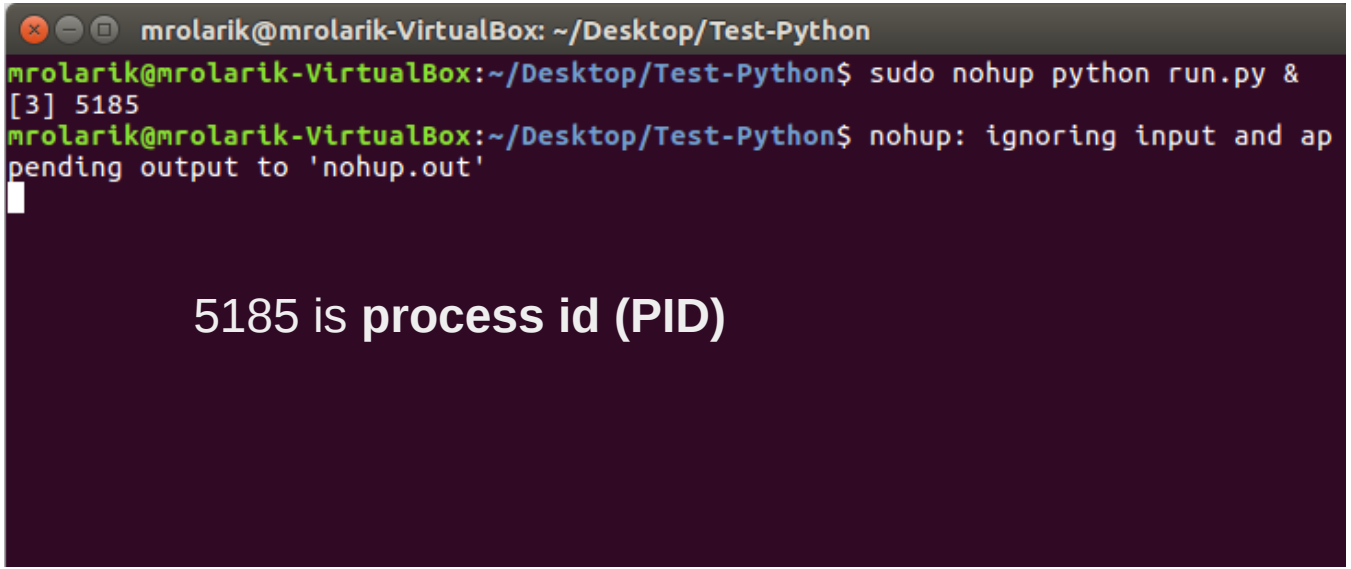
$ nohup any-program &

- nohup is often used in combination with the “**nice**” command to run processes on a lower priority.

$ nohup nice any-program &

#nohup

$ sudo nohup python run.py &

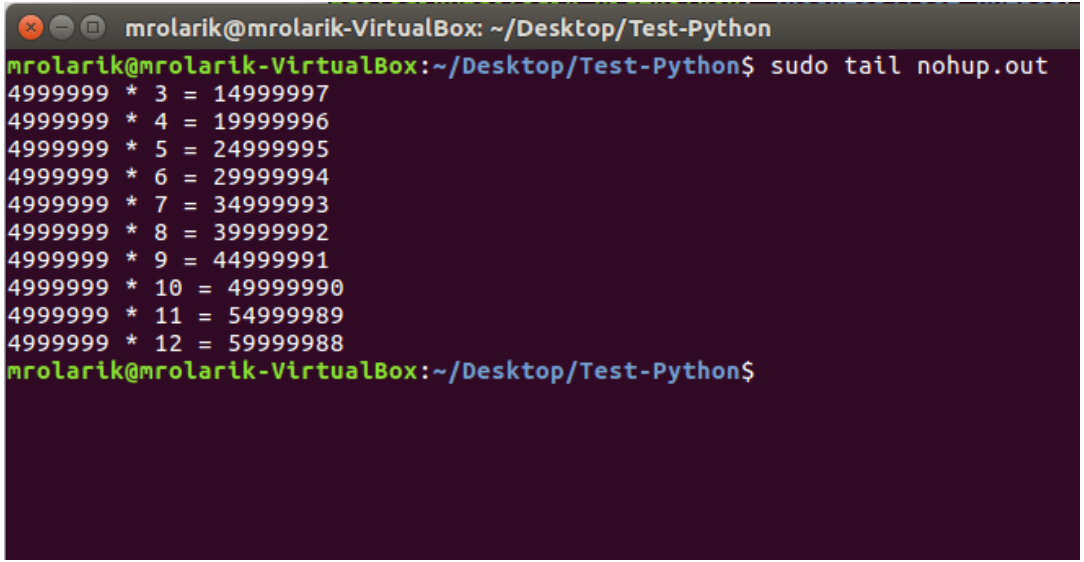


#tail

$ sudo tail -f nohup.out

$ sudo tail -f show.txt

ดูผลลัพธ์ของโปรแกรม



References

- https://www.tutorialspoint.com/python/python_basic_syntax.htm
- https://docs.python.org/2/tutorial/introduction.html#using-python-as-a-calculator
- http://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/functions.html
- https://www.tutorialspoint.com/python/python_functions.htm
- http://unix.stackexchange.com/questions/165214/how-to