

# 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
```

```
$ sudo nano if-test.py
```

```
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")
```

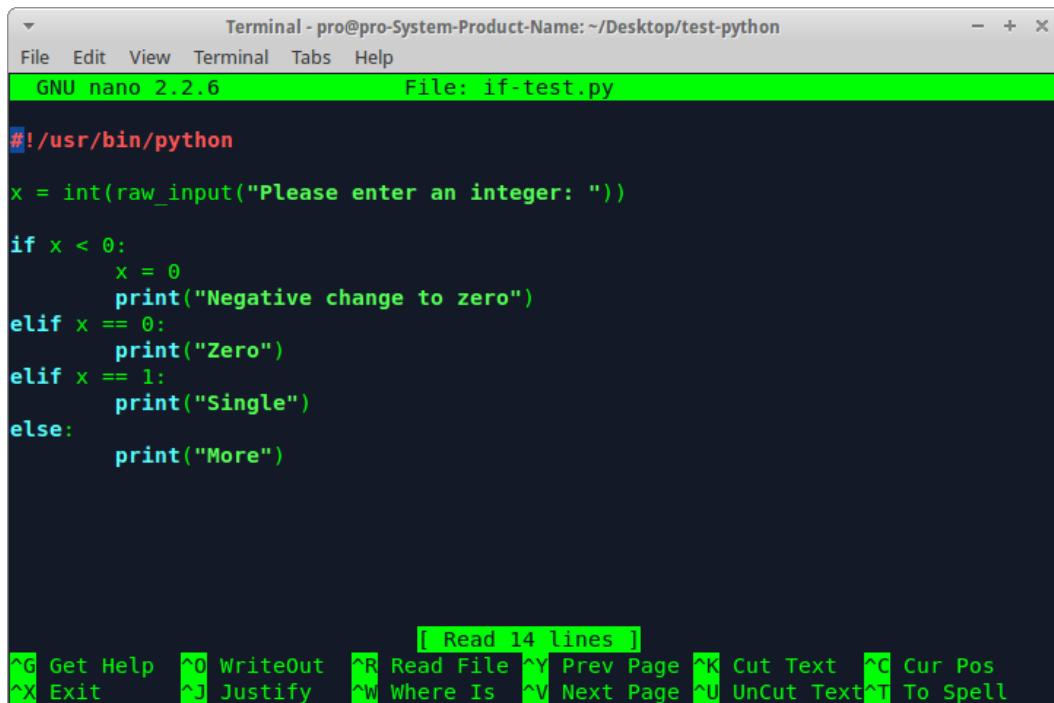
```
else:
```

```
    print("More")
```

# Control Flow: if Statements

```
$ chmod +x if-test.py
```

```
$ ./if-test.py
```



The screenshot shows a terminal window titled "Terminal - pro@pro-System-Product-Name: ~/Desktop/test-python". The window has a green header bar with the text "GNU nano 2.2.6" and "File: if-test.py". The main area of the terminal displays the following Python code:

```
#!/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")
```

At the bottom of the terminal window, there is a status bar with the message "[ Read 14 lines ]". Below the status bar, there is a menu bar with options: "File Edit View Terminal Tabs Help". At the very bottom, there is a row of keyboard shortcuts:

- ^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

# 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	$\leq$	$\leq$
Greater than or equal	$\geq$	$\geq$
Equals	=	==
Not equal	$\neq$	!=

# if statements

```
score = float(input('Enter \nexamination 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 ^I 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.'''\n\nPI = 3.14159265358979      # global constant -- only place the value of PI is set\n\ndef circleArea(radius):\n    return PI*radius*radius      # use value of global constant PI\n\ndef circleCircumference(radius):\n    return 2*PI*radius          # use value of global constant PI\n\ndef main():\n    print('circle area with radius 5:', circleArea(5))\n    print('circumference with radius 5:', circleCircumference(5))\n\nmain()
```

^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 ^I 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.'''\n\nPI = 3.14159265358979      # global constant -- only place the value of PI is set\n\ndef circleArea(radius):\n    return PI*radius*radius      # use value of global constant PI\n\ndef circleCircumference(radius):\n    return 2*PI*radius          # use value of global constant PI\n\ndef main():\n    print('circle area with radius 5:', circleArea(5))\n    print('circumference with radius 5:', circleCircumference(5))\n\nmain()
```

^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 is incorrect - try again")
```

# Check if raw input is integer

```
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 integer

```
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 {}".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()
```

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 ^I 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.'''\n\nPI = 3.14159265358979      # global constant -- only place the value of PI is set\n\ndef circleArea(radius):\n    return PI*radius*radius      # use value of global constant PI\n\ndef circleCircumference(radius):\n    return 2*PI*radius          # use value of global constant PI\n\ndef main():\n    print('circle area with radius 5:', circleArea(5))\n    print('circumference with radius 5:', circleCircumference(5))\n\nmain()
```

^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

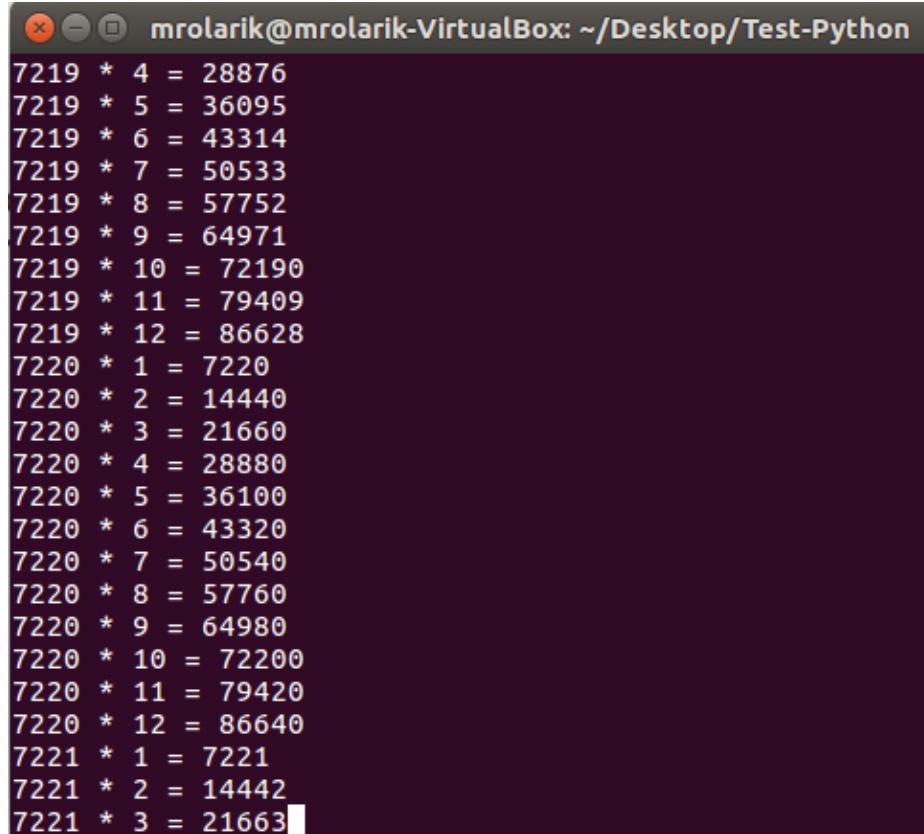
file = open('show.txt','w')
for x in xrange(1,5000000):
    for y in xrange(1,13):
        print '%d * %d = %d' %(x,y,x*y)
        file.write('%d * %d = %d\n' %(x,y,x*y))

file.close()
```

# #testing “run.py” program

**\$ python run.py**

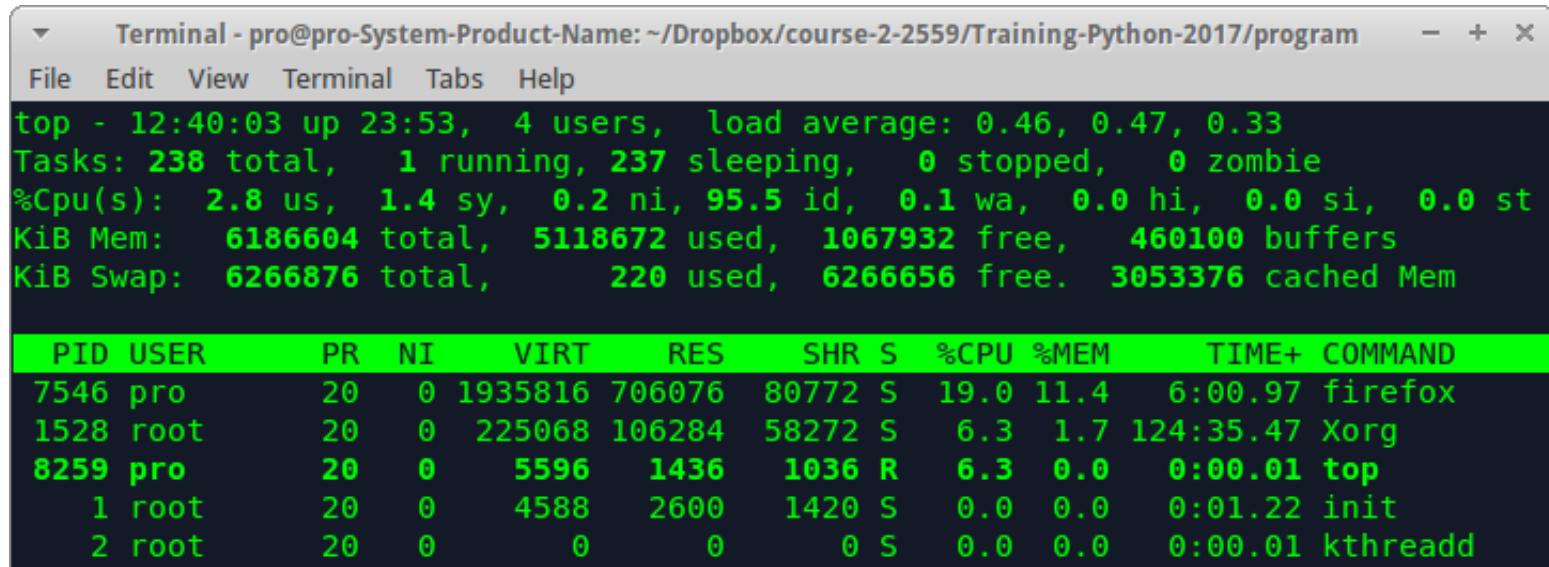
# the process will  
**end** when you  
Close the terminal



```
molarik@molarik-VirtualBox: ~/Desktop/Test-Python
7219 * 4 = 28876
7219 * 5 = 36095
7219 * 6 = 43314
7219 * 7 = 50533
7219 * 8 = 57752
7219 * 9 = 64971
7219 * 10 = 72190
7219 * 11 = 79409
7219 * 12 = 86628
7220 * 1 = 7220
7220 * 2 = 14440
7220 * 3 = 21660
7220 * 4 = 28880
7220 * 5 = 36100
7220 * 6 = 43320
7220 * 7 = 50540
7220 * 8 = 57760
7220 * 9 = 64980
7220 * 10 = 72200
7220 * 11 = 79420
7220 * 12 = 86640
7221 * 1 = 7221
7221 * 2 = 14442
7221 * 3 = 21663
```

# top command

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



The screenshot shows a terminal window with the following details:

- Terminal title: Terminal - pro@pro-System-Product-Name: ~/Dropbox/course-2-2559/Training-Python-2017/program
- Menu bar: File, Edit, View, Terminal, Tabs, Help
- Output of the top command:

```
top - 12:40:03 up 23:53, 4 users, load average: 0.46, 0.47, 0.33
Tasks: 238 total, 1 running, 237 sleeping, 0 stopped, 0 zombie
%Cpu(s): 2.8 us, 1.4 sy, 0.2 ni, 95.5 id, 0.1 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem: 6186604 total, 5118672 used, 1067932 free, 460100 buffers
KiB Swap: 6266876 total, 220 used, 6266656 free. 3053376 cached Mem
```
- Process table header:

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
-----	------	----	----	------	-----	-----	---	------	------	-------	---------

- Process table data:

7546	pro	20	0	1935816	706076	80772	S	19.0	11.4	6:00.97	firefox
1528	root	20	0	225068	106284	58272	S	6.3	1.7	124:35.47	Xorg
8259	pro	20	0	5596	1436	1036	R	6.3	0.0	0:00.01	top
1	root	20	0	4588	2600	1420	S	0.0	0.0	0:01.22	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.01	kthreadd

# 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

```
Terminal - pro@pro-System-Product-Name: ~/Dropbox/course-2-2559/Training-Python-2017/program
File Edit View Terminal Tabs Help
41401*4 = 165604
41401*5 = 207005
41401*6 = 248406
41401*7 = 289807
41401*8 = 331208
41401*9 = 372609
41401*10 = 414010
41401*11 = 455411
41401*12 = 496812
41402*1 = 41402
41402*2 = 82804
41402*3 = 124206
41402*4 = 165608
41402*5 = 207010
41402*6 = 248412
41402*7 = 289814
41402*8 = 331216
41402*9 = 372618
41402*10 = 414020
41402*11 = 455422
41402*12 = 496824
41403*1 = 41403Terminated
pro@pro-System-Product-Name:~/Dropbox/course-2-2559/Training-Python-2017/program
$
```

# 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

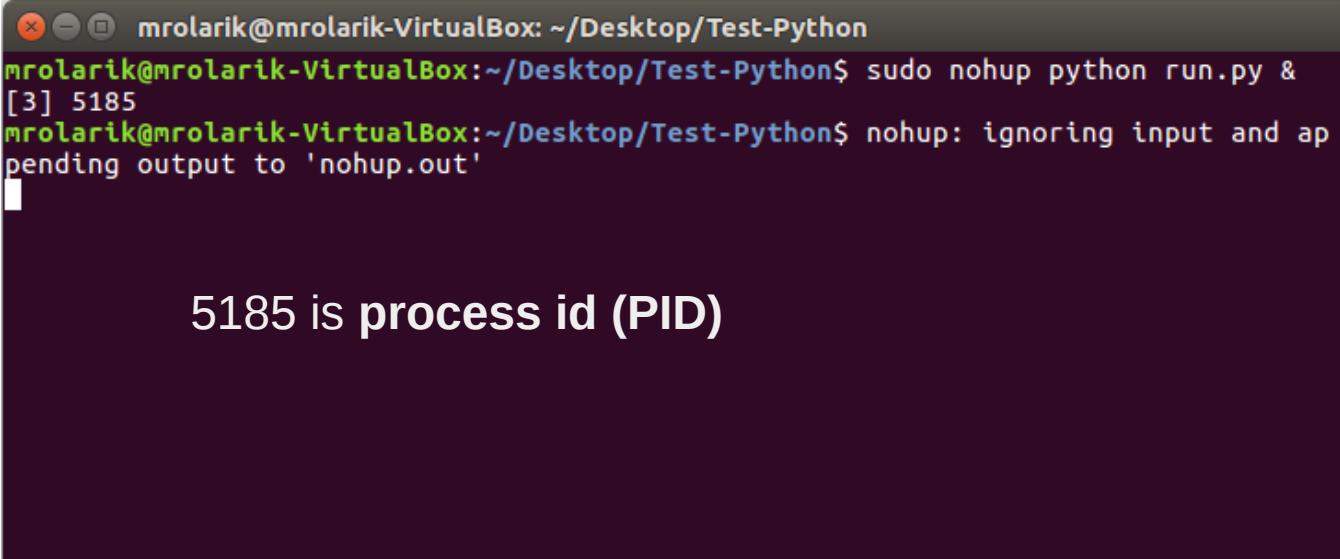
```
$ nohup any-program &
```

- 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 &
```



The screenshot shows a terminal window with the following text:

```
molarik@molarik-VirtualBox:~/Desktop/Test-Python
molarik@molarik-VirtualBox:~/Desktop/Test-Python$ sudo nohup python run.py &
[3] 5185
molarik@molarik-VirtualBox:~/Desktop/Test-Python$ nohup: ignoring input and ap
pending output to 'nohup.out'
```

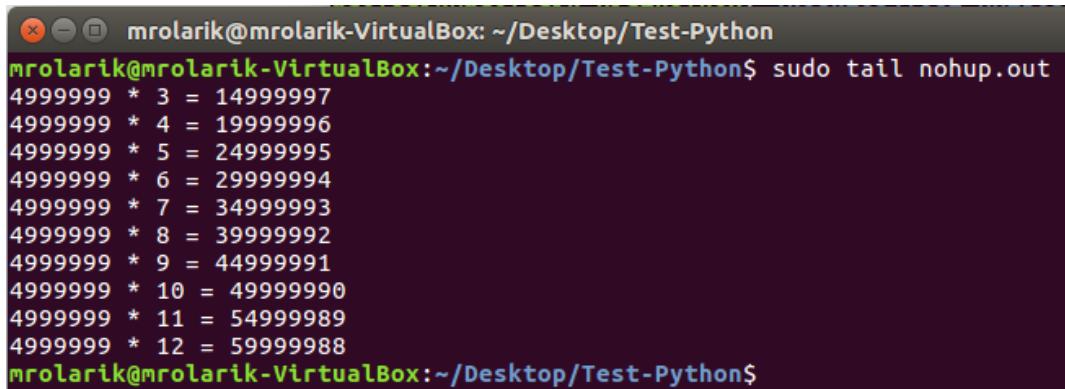
5185 is process id (PID)

# #tail

\$ sudo tail -f nohup.out

\$ sudo tail -f show.txt

# ดูผลลัพธ์ของโปรแกรม



The screenshot shows a terminal window titled "molarik@molarik-VirtualBox: ~/Desktop/Test-Python". The command "sudo tail nohup.out" is run, and the output is displayed. The output consists of a series of multiplication results where 4999999 is multiplied by integers from 3 to 12, followed by a final empty line. The terminal prompt "molarik@molarik-VirtualBox:~/Desktop/Test-Python\$" is visible at the bottom.

```
molarik@molarik-VirtualBox:~/Desktop/Test-Python$ sudo tail nohup.out
4999999 * 3 = 14999997
4999999 * 4 = 19999996
4999999 * 5 = 24999995
4999999 * 6 = 29999994
4999999 * 7 = 34999993
4999999 * 8 = 39999992
4999999 * 9 = 44999991
4999999 * 10 = 49999990
4999999 * 11 = 54999989
4999999 * 12 = 59999988
molarik@molarik-VirtualBox:~/Desktop/Test-Python$
```

# References

- [https://www.tutorialspoint.com/python/python\\_basic\\_syntax.htm](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](https://www.tutorialspoint.com/python/python_functions.htm)
- <http://unix.stackexchange.com/questions/165314/how-to>