FireTech Camp Python Cheat Sheet

Getting Started

Starting IDLE

- Microsoft Windows: Start → All Programs → Python 3.3 → IDLE
- Apple Mac OSX: Applications → Python 3.3 → IDLE
- Linux: Terminal → “idle”

Checking Version

```
import sys
print(sys.version) #x.y.z
```

Major number (x, the first one) should be 3

Output

```
print('Hello, World!') #Hello, World!
```

Input

```
input_string = input('Prompt text here')
print(input_string) #whatever the user typed
```

Variables

Assignment

```
my_variable = 123
print(my_variable) #123
```

The variable name (*my_variable*) can be any combination of numbers or letters, but must start with a letter
The variable value (*value*) can any valid value or variable name

Types/Conversions

```
variable_name = 'test'
print(type(variable_name)) #<type 'str'>
```

The type function returns a string of the current type

```
my_variable = 123
my_string_variable = str(my_variable)
print(my_string_variable) #'123'
```

Primitive types:

Type /Cast Function	Name	Example	String Symbol
bool	Boolean	True,False	%r
int	Integer	-1,-2,0,3,6,82	%d
float	Whole Number	-2.3,0.0,65.2,7	%f
str	String	'test','1','a phrase'	%s

String Formatting

```
my_var = '%s\t%f' % ('a string',1.0)
print('%s' % (my_var)) #a string        1.0
```

Formatting Codes:

Symbol	Name	Meaning
%n	New line	Starts a new line
%t	Tab	Moves to next indentation tab
%{String Symbol}	String Formatting	Creates a formatted string

Arithmetic Operations

```
answer = 1 + 1
print(answer) #2
```

Arithmetic operations supported in precedence order:

Symbols	Notes
()	Use this to ensure operations in the brackets are evaluated first
**	Exponential Operator
*,/,//,%	// is integer division, % is remainder
+,-	
=, +=, -=, *=, /=	This performs the operation on the current value and assigns it

Logic

```
var1 = 123
var2 = 123
answer = (var1 == var2)
print(answer) #True
```

Boolean Operators:

Symbols	Use
==,! =	Equals and not Equals
<,>	Less and more than
<=,>=	equals or less and more than

Logical Operators

```
var1 = True
var2 = (123 == 124) #False
answer = var1 and var2
print(answer) #False
```

Logical Operators:

Symbols	Use
and, &&	are both true?
or, ||	is either one true?

Control

Conditionals

```
if(var1==var2):
    print('They're equal!')
elif(var1>var2):
    print('var1 is bigger')
else:
    print('they're not equal!')
```

Note the indentation!

Loops

```
var1 = 0
var2 = 10
while(var1<var2):
    print(var1) #0 ... 9
    var1 += 1
```

Data Structures

Lists

```
my_list = ['abc','def','ghi']
print(my_list[1]) #def
```

The contents of the list can be any mixture of valid variables and constant values.

```
my_list = ['abc','def','ghi','jkl']
for letters in my_list:
        print(letters) #abc ... jkl
```

```
for i in range(10):
        print(i) #0 ... 9
```

This is functionally equivalent to the while loop example.

Dictionaries

```
my_dict = {'abc':123,'def':456,'ghi':789}
print(my_dict['ghi']) #789
```

```
my_dict = {'abc':123,'def':456,'ghi':789}
for key in my_dict.keys():
        print(my_dict[key]) #abc ... ghi
```

Files

```
my_writefile = open('file_test.txt','w')
my_writefile.write('123')
my_writefile.close()
my_readfile = open('file_test.txt','r')
print(my_readfile.read()) #123
my_readfile.close()
```

Creates a file called read_test.txt exists in the current working directory.

Reusable Code

Functions

```
def my_function(arg1,arg2):
        print(arg1) #x
        print(arg2) #123

my_function('x',123)
```

Classes

```
class my_class:
        attribute = None

        def __init__(self,arg1):
                self.attribute = arg1

        def my_method(self):
                print self.attribute

my_instance = my_class('123')
my_instance.my_method() #123
```

```
class my_child_class(my_class):
        attribute2 = None

        def __init__(self,arg1,arg2):
                my_class.__init__(self,arg1)
                self.attribute2 = arg2

        def my_method(self):
                my_class.my_method(self)
                print self.attribute2

my_instance = my_child_class('hi',123)
my_instance.my_method() #hi, then 123
```

Note assuming the previous code snippet

Installing 3rd Party Libraries

Using PIP

1.
   - Microsoft Windows: Start → All Programs → Administrative Command Prompt
   - Apple Mac OSX/Linux: Open Terminal
2.
   - Microsoft Windows: 'pip install <package name>'
   - Apple Mac OSX/Linux: 'sudo pip install <package name>'

   <package name> is the name of the package you want to install

Using distutils

* **NB** only use this method with files from a trusted source *

1. Download and unpack the library to a location of your choice
2.
   - Microsoft Windows: Start → All Programs → Administrative Command Prompt
   - Apple Mac OSX/Linux: Open Terminal
3.
   - Microsoft Windows: 'cd <unpacked package location>' and press enter
   - Apple Mac OSX/Linux: type 'cd <unpacked package location>' and press enter

   <unpacked package locate> is the location of the package you want to install
4.
   - Microsoft Windows: type 'python setup.py install' and press enter
   - Apple Mac OSX/Linux: type 'sudo python setup.py install' and press enter. You will have to enter the your password and you must have administrative privileges.

Kivy

Installing

Download and install from `http://kivy.org/#download`

Running

- Microsoft Windows:
  1. Start → All Programs → Administrative Command Prompt
  2. type 'cd <code location>' (where <code location> is the location of your code
  3. type 'kivy <your code name>.py' and press enter
- Apple Mac OSX/Linux:
  1. Open Terminal
  2. type 'cd <code location>' (where <code location> is the location of your code
  3. type 'kivy <your code name>.py' and press enter

Building

```
from kivy.app import App
from kivy.uix.label import Label

class my_app(App):
        root = None

        def build(self):
                return self.root()

app = my_app()
label = Label(text='Text')
app.root = label
app.run()
```

Buttons and Popups

```
from kivy.uix.button import Button
from kivy.uix.popup import Popup

def my_bcallback(instance):
        popup = Popup(title='Button Popup',
                content=Label(text='Text'),
                size=(400, 400))
        popup.open()

app = my_app()
button = Button(text='Button')
button.bind(on_press=my_bcallback)
app.root = button
app.run()
```

Text Input

```
from kivy.app import App
from kivy.uix.textinput import TextInput

text = None
def my_icallback(instance):
        global text
        text = instance._get_text()
        popup = Popup(title=TextInput Popup',
                content=Label(text=text),
                size=(400, 400))
        popup.open()

app = MyApp()
textinput = TextInput(text=Input Text,
                        multiline=False)
textinput.bind(on_text_validate=my_icallback)
app.root = textinput
app.run()
```

Grid Layouts

Requires the functions and imports from the previous code snippets

```
from kivy.uix.gridlayout import GridLayout

app = MyApp()
textinput = TextInput(text=Input Text,
                multiline=False)
textinput.bind(on_text_validate=My_icallback)
button = Button(text=Button')
button.bind(on_press=my_bcallback)

layout = GridLayout(cols=1)
layout.add_widget(button)
layout.add_widget(textinput)
app.root = layout
app.run()
```