Basic Python Workshop

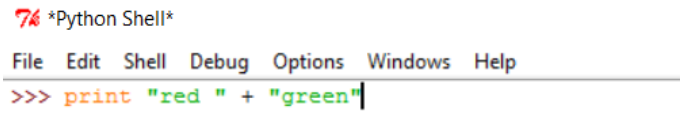
Colorado School of Mines

Summer 2016

IDLE Exploration

IDLE Exploration: Investigation

1. **To open IDLE: Search for IDLE in Windows Explorer (It may take a few seconds to load)**
2. **Running a basic command from the Python Shell**
   a) Type color = “red” and then press return
   b) Then type color and press return.
   c) In the Python Shell, type a command to print the strings “red” and “green” as shown below.



3. **Write a function definition in the Python Shell and call the function from the Python Shell**
   a) In the Python Shell, write the following function definition and then run the script by pressing return twice.

```
>>> def printColor(theColor):
        print theColor + "red"
```

   b) Now, call printColor(“pink”) by typing it below the function. What happens?
   c) Try to alter the printColor function definition. Is it possible?

4. **Write a function definition in a Python File and call the function from the Python Shell.**
   a) From the Python Shell, click File -> New Window.
   b) You should then see a new window appear. In this new window, click File -> Save As
   c) Save the file as “IdleIntro.py”
   d) Write the following function definition in the IdleIntro.py file

```
def printColor2(color1, color2):
    print "my favorite colors are " + color1 + color2
```

   e) Run the IndleIntro file script by selecting Run->Run Module from the bar at the top of the screen or by pressing F5.
   f) In the Python Shell call the printColor2 function and pass it 2 string parameters.

```
>>> printColor2("yellow", "green")
```

   g) Press enter to run.

5. Write a function definition in a Python file and call the function from the Python file.

It is also possible to call a function from within a python file. We will call the printColor2 function from within the IdleIntro.py file.

a) Call printColor2 from below the printColor2 function definition.

```
def printColor2(color1, color2):
    print "my favorite colors are " + color1 + color2

printColor2("red", "green")
```

b) Save the IdleIntro.py file and then Run the file by selecting Run->Run Module or by pressing F5. Notice where the output is printed.

IDLE Exploration: Activity

Write a function that intakes a value and converts the value from Fahrenheit to Celsius.

You must:

a) **Write the function definition in a new file, not the Python Shell**
b) **Call the function from the Python Shell.**

The conversion formula for Celsius -> Farenheit is:

$$F = 32 + \frac{5}{9}C$$

Rock Paper Scissors

Use this sheet to document your brainstorming process. Use the notes section to keep track of questions/comments for group discussion or thoughts about classroom implementation.

Notes

Tracing Code

Code Tracing Example:

```
#!/usr/bin/python

from random import randint

def make_guess(lowNum, highNum):
    global total_guesses
    total_guesses += 1
    print "Guessing between:", lowNum, "and", highNum
    return lowNum + ((highNum - lowNum) / 2)

low = 0
high = 100

MAX_GUESSES = 8
total_guesses = 0

print "Think of a number between " + str(low) + " and " + str(high

guess = make_guess(low, high)
response = raw_input("Is " + str(guess) + " the number (c)? Or High (h)? or Low (l)? ")

while response.lower() != 'c' and total_guesses < MAX_GUESSES:
    if response.lower() == 'h':
        high = guess
    elif response.lower() == 'l':
        low = guess
    else:
        print "That wasn't a valid response. Try again..."
        response = raw_input("Is " + str(guess) + " the number (c)? Or High (h)? or Low (l)? ")
        continue
    guess = make_guess(low, high)
    response = raw_input("Is " + str(guess) + " the number (c)? Or High (h)? or Low (l)? ")

if response.lower() == 'c':
    print "I guessed the correct number in " + str(total_guesses) + " guesses!"
else:
    print "I took more than " + str(MAX_GUESSES) + "...so I lose"
```

Buggy Code:

```
from random import randint

low = 0
high = 512
number = randint(low, high)

guess = -1
total_guesses = 0

prompt = "Try to guess the number (between "  str(low)  " and "  str(high)  "): "

while guess = number:

    guess = input(prompt)
    total_guesses = 1
    if guess < number:
        print "Too High, Try Again"
    else guess > number:
        print "Too Low, Try Again"

    print "You got it! The number was", number
    print "It took you", total_guesses, "guesses to guess the number."
```

Turtle Art

Code included in zip file

```
import turtle
import math

# Draws n line segments.
def p_line(t, n, length, angle):
    for i in range(n):
        t.fd(length)
        t.lt(angle)

# Draws a polygon with n sides.
def polygon(t, n, length):
    angle = 360/n
    p_line(t, n, length, angle)

# Draws an arc with the given radius and angle.
def arc(t, r, angle):
    arc_length = 2 * math.pi * r * abs(angle) / 360
    n = int(arc_length / 4) + 1
    step_length = arc_length / n
    step_angle = float(angle) / n

    # Before starting reduces, making a slight left turn.
    t.lt(step_angle/2)
    p_line(t, n, step_length, step_angle)
    t.rt(step_angle/2)

# Draws a petal using two arcs.
def petal(t, r, angle):
    for i in range(2):
        arc(t, r, angle)
        t.lt(180-angle)

# Draws a flower with n petals.
def flower(t, n, r, angle, p):
    for i in range(n):
        petal(t, r, angle)
        t.lt(p/n)

# Draws a leaf and fill it.
def leaf(t, r, angle, p):
```

```
    t.begin_fill() # Begin the fill process.
    t.down()
    flower(t, 1, 40, 80, 180)
    t.end_fill()

# Now draw the flower using the helper functions

def main():
    window=turtle.Screen() #create a screen
    window.bgcolor("blue")
    lucy=turtle.Turtle()
    lucy.shape("turtle")
    lucy.color("red")
    lucy.width(5)
    lucy.speed(0)

    # Drawing flower
    flower(lucy, 10, 40, 100, 360)

    # Drawing stem
    lucy.color("brown")
    lucy.rt(90)
    lucy.fd(200)

    # Drawing leaf
    lucy.rt(270)
    lucy.color("green")
    leaf(lucy, 40, 80, 180)
    lucy.ht()

    window.exitonclick()

main()  # call the main function
```

Mastermind

The rules of Mastermind:

- Mastermind is a game with 2 roles:
    - Code maker
    - Code breaker
- The code maker selects a secret 4 digit number
- The code breaker guesses a 4 digit number
- The code maker tells the code breaker how many digits of the guessed number are present and how many numbers are in the correct place.
- Play continues until all 4 numbers are correct and in the correct location

Use the space below to brainstorm:

Notes

Bins Distribution

- **Task 1:** Write a script to randomly generate a file of integer values
- **Task 2:** Partition the data in a way of your choice. (0 - 10, 10 - 20 …. 90 - 100)
    - Show a horizontal histogram of your data

Use the space below to brainstorm:

Notes

Game of Sticks

Rules:

1. Players start with a group of 10 – 15 objects
2. Player 1 takes 1, 2 or 3 of the objects
3. Player 2 takes 1, 2 or 3 of the objects
4. Play continues in this fashion until a player is forced to take the last object

Use the space below to brainstorm:

Notes

Python Syntax Cheat Sheet

Keywords:

- `and`
- `as`
- `assert`
- `break`
- `class`
- `continue`
- `def`
- `del`
- `elif`
- `else`
- `except`
- `exec`
- `False`
- `finally`
- `for`
- `from`
- `global`
- `if`
- `import`
- `in`
- `is`
- `lambda`
- `None`
- `nonlocal`
- `not`
- `or`
- `pass`
- `print`
- `raise`
- `return`
- `True`
- `try`
- `while`
- `with`
- `yield`

Python quick reference sheet

Python 2.7 Quick Reference Sheet

ver 2.01 – 110105 (sjd)

Interactive Help in Python Shell

help()	Invoke interactive help
help(*m*)	Display help for module *m*
help(*f*)	Display help for function *f*
dir(*m*)	Display names in module *m*

Small Operator Precedence Table

func_name(*args, ...*)	Function call
x[*index* : *index*]	Slicing
x[*index*]	Indexing
x.attribute	Attribute reference
**	Exponentiation
*, /, %	Multiply, divide, mod
+, -	Add, subtract
>, <, <=, >=, !=, ==	Comparison
in, not in	Membership tests
not, and, or	Boolean operators NOT, AND, OR

Module Import

```
import module_name
from module_name import name , ...
from module_name import *
```

Common Data Types

Type	Description	Literal Ex
int	32-bit Integer	3, -4
long	Integer > 32 bits	101L
float	Floating point number	3.0, -6.55
complex	Complex number	1.2J
bool	Boolean	True, False
str	Character sequence	"Python"
tuple	Immutable sequence	(2, 4, 7)
list	Mutable sequence	[2, x, 3.1]
dict	Mapping	{ x:2, y:5 }

Common Syntax Structures

Assignment Statement
```
var = exp
```

Console Input/Output
```
var = input( [prompt] )
var = raw_input( [prompt] )
print exp[,] ...
```

Selection
```
if (boolean_exp):
   stmt ...
[elif (boolean_exp):
   stmt ...] ...
[else:
   stmt ...]
```

Repetition
```
while (boolean_exp):
   stmt ...
```

Traversal
```
for var in traversable_object:
   stmt ...
```

Function Definition
```
def function_name( parmameters ):
   stmt ...
```

Function Call
```
function_name( arguments )
```

Class Definition
```
class Class_name [ (super_class) ]:
   [ class variables ]
   def method_name( self,  parameters ):
      stmt
```

Object Instantiation
```
obj_ref = Class_name( arguments )
```

Method Invocation
```
obj_ref.method_name( arguments )
```

Exception Handling
```
try:
   stmt ...
except [exception_type] [, var]:
   stmt ...
```

Common Built-in Functions

Function	Returns
abs(*x*)	Absolute value of *x*
dict()	Empty dictionary, eg: d = dict()
float(*x*)	int or string *x* as float
id(*obj*)	memory addr of *obj*
int (*x*)	float or string *x* as int
len(*s*)	Number of items in sequence *s*
list()	Empty list, eg: m = list()
max(*s*)	Maximum value of items in *s*
min(*s*)	Minimum value of items in *s*
open(*f*)	Open filename *f* for input
ord(*c*)	ASCII code of *c*
pow(*x,y*)	x ** y
range(*x*)	A list of x ints 0 to *x* - 1
round(*x,n*)	float *x* rounded to *n* places
str(*obj*)	str representation of *obj*
sum(*s*)	Sum of numeric sequence *s*
tuple(*items*)	tuple of *items*
type(*obj*)	Data type of *obj*

Common Math Module Functions

Function	Returns (all float)
ceil(*x*)	Smallest whole nbr >= *x*
cos(*x*)	Cosine of *x* radians
degrees(*x*)	*x* radians in degrees
radians(*x*)	*x* degrees in radians
exp(*x*)	e ** *x*
floor(*x*)	Largest whole nbr <= *x*
hypot(*x, y*)	sqrt(*x* * *x* + *y* * *y*)
log(*x* [, *base*])	Log of *x* to *base* or natural log if *base* not given
pow(*x, y*)	x ** y
sin(*x*)	Sine of *x* radians
sqrt(*x*)	Positive square root of *x*
tan(*x*)	Tangent of *x* radians
pi	Math constant pi to 15 sig figs
e	Math constant e to 15 sig figs

Common String Methods

***S*.method()**	**Returns (str unless noted)**
capitalize	*S* with first char uppercase
center(*w*)	*S* centered in str *w* chars wide
count(*sub*)	int nbr of non-overlapping occurrences of *sub* in *S*
find(*sub*)	int index of first occurrence of *sub* in *S* or -1 if not found
isdigit()	bool True if *S* is all digit chars, False otherwise
islower() isupper()	bool True if *S* is all lower/upper case chars, False otherwise
join(*seq*)	All items in *seq* concatenated into a str, delimited by *S*
lower() upper()	Lower/upper case copy of *S*
lstrip() rstrip()	Copy of *S* with leading/ trailing whitespace removed, or both
split([*sep*])	List of tokens in *S*, delimited by *sep*; if *sep* not given, delimiter is any whitespace

Formatting Numbers as Strings

Syntax: *"format_spec" % numeric_exp*
***format_spec* syntax:** *% width.precision type*

- *width* (optional): align in number of colums specified; negative to left-align, precede with 0 to zero-fill
- *precision* (optional): show specified digits of precision for floats; 6 is default
- *type* (required): d (decimal int), f (float), s (string), e (float – exponential notation)
- Examples for x = 123, y = 456.789
  "%6d" % x -> . . . 123
  "%06d" % x -> 000123
  "%8.2f % y -> . . 456.79
  "8.2e" % y -> 4.57e+02
  "-8s" % "Hello" -> Hello . . .

Common List Methods

***L*.method()**	**Result/Returns**
append(*obj*)	Append *obj* to end of *L*
count(*obj*)	Returns int nbr of occurrences of *obj* in *L*
index(*obj*)	Returns index of first occurrence of *obj* in *L*; raises ValueError if *obj* not in *L*
pop([*index*])	Returns item at specified *index* or item at end of L if *index* not given; raises IndexError if *L* is empty or *index* is out of range
remove(*obj*)	Removes first occurrence of *obj* from *L*; raises ValueError if *obj* is not in *L*
reverse()	Reverses *L* in place
sort()	Sorts *L* in place

Common Tuple Methods

***T*.method()**	**Returns**
count(*obj*)	Returns nbr of occurrences of *obj* in *T*
index(*obj*)	Returns index of first occurrence of *obj* in *T*; raises ValueError if *obj* is not in *T*

Common Dictionary Methods

***D*.method()**	**Result/Returns**
clear()	Remove all items from *D*
get(*k* [,*val*])	Return *D*[*k*] if *k* in *D*, else *val*
has_key(*k*)	Return True if *k* in *D*, else False
items()	Return list of key-value pairs in *D*; each list item is 2-item tuple
keys()	Return list of *D*'s keys
pop(*k*, [*val*])	Remove key *k*, return mapped value or *val* if *k* not in *D*
values()	Return list of *D*'s values

Common File Methods

***F*.method()**	**Result/Returns**
read([*n*])	Return str of next *n* chars from *F*, or up to EOF if *n* not given
readline([*n*])	Return str up to next newline, or at most *n* chars if specified
readlines()	Return list of all lines in *F*, where each item is a line
write(*s*)	Write str *s* to *F*
writelines(*L*)	Write all str in seq *L* to *F*
close()	Closes the file

Other Syntax

Hold window for user keystroke to close:
raw_input("Press <Enter> to quit.")

Prevent execution on import:
if __name__ == "__main__":
main()

Displayable ASCII Characters

32	SP	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	105	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	DEL

'\0' = 0, '\t' = 9, '\n' = 10

Reading from a file

```
#Read through line by line
shirtsfile=open('shirts.txt', 'r')

#Do something with each line
for line in shirtsfile.readlines():
    print line
```

```
#Read one line from a file with readlines()

shirtsfile = open('shirts.txt', 'r')
shirts = shirtsfile.readlines()
print shirts
```

```
#Read one line from a file with read().splitlines()

shirtsfile = open('shirts.txt', 'r')
shirts = shirtsfile.read().splitlines()
print shirts
```

Writing to a file

```
theFile = open('numFile.txt', 'w')

for i in range(0, 3):
    theFile.write(str(i) + "\n")

theFile.close()
```