Programs, Variables & Expressions

Programming (for biologists)

BIOL 7800

Operators

In **python**, these are the common (arithmetic) operators

+	-	*	/	//	%	**
addition	subtraction	multiplication	division	integer div.	modulus	exponent

Order of Operations

PEMDAS

Parentheses are executed first

Exponents are executed second

Mutiplication and **D**ivision are executed third

Addition and **S**ubtraction are executed fourth

Operators with the same precedence are run from **left** to **right**

Assignment statements

equals vs. equals-equals

= **vs.** **==**

Assignment

"set variable to something"

e.g. **x = 5**

Equivalence

"this is equal to that"

e.g.

```
if x == 5:
    do something
```

Variable naming

Variable names are typically lowercase, descriptive, and separated by underscores

(spaces are not allowed)

```
my_variable_name = 10
```

In other languages, variables are often "camel-cased" AKA CapWords

```
MyVariableName = 10
```

(This naming scheme is suggested for python classes)

Values and types

Python is an

object-oriented, interpreted, garbage-collected language

```python
def _mean_and_variance_pop_n(values):
    n = 0
    s = 0.0
    ss = 0.0
    for v in values:
        n += 1
        s += v
        ss += v*v
    if n == 0:
        raise IndexError("values in mean_and_variance cannot be empty")
    mean = float(s)/n
    var = (ss - mean*s)/n
    return mean, var, n
```

Very high-level language

dynamically typed

Values and types

But, just because Python is dynamically typed
does not mean variables don't have a type

`type()` function

`type(100)`	`type(2.6)`	`type("cat")`
`Out[3]: int`	`Out[4]: float`	`Out[5]: str`

What about:

`type("2.6")`

??

Expressions v. Statements

Just some programming nomenclature...

expressions are combinations of
values, variables, and operators

```
n = 12
n + 24 * 36
```

statements are "units of code"
that have some effect

```
print("dog")
type(2.6)
```

String Operations

Can you add a string?

"brrrrrp" **"is a"** **"onomatopoeia"**

Why, yes, <u>you can</u>!

"brrrrrp" + "is a" + "onomatopoeia"

"brrrrrpis aonomatopoeia"

This is known as "string concatenation"

String Operations

But what about multiplication?

```
"brrrrrrp" * 10
```

```
Out[6]: 'brrrrrrpbrrrrrrpbrrrrrrpbrrrrrrpbrrrrrrpbrrrrrrpbrrrrrrpbrrrrrrpbrrrrrrpbrrrrrrp'
```

Comments

There are several kinds of comments you will see in Python programs

standard comment above something
followed by the something you are commenting

Comments

There are several kinds of comments you will see in Python programs

code you are commenting # followed by a comment

(these are called "in-line" comments)

Comments

There are several kinds of comments you will see in Python programs

```
'''
here is a giant block of code
that you would like to make
a giant comment about.

This is also used for preambles,
license info, novels, etc.
'''
```

(these are called "block" comments)

Comments

Generally, comments should be informative and document non-obvious parts of the code

This is redundant

```
# set variable dogs to "stinky"
dogs = "stinky"
```

This is better, but still somewhat redundant

```
# get values 0 to 50 by 5
[elem for elem in range(0,51) if elem % 5 == 1]
```

Debugging

We will get to "fancy" debugging, but one of the first debugging "tools" to use is the **print()** function

```
for number in range(0, 100):
    if number % 5 == 0:
        my_special_function(number)
```

Debugging

We will get to "fancy" debugging, but one of the first debugging "tools" to use is the **print()** function

```
for number in range(0, 100):
    if number % 5 == 0:
        print(number)
        my_special_function(number)
```

Debugging

We will get to "fancy" debugging, but one of the first debugging "tools" to use is the **print()** function

```
for number in range(0, 100):
    if number % 5 == 0:
        my_special_function(number)
    else:
        print(number)
```

Programs

How to run them

2. python

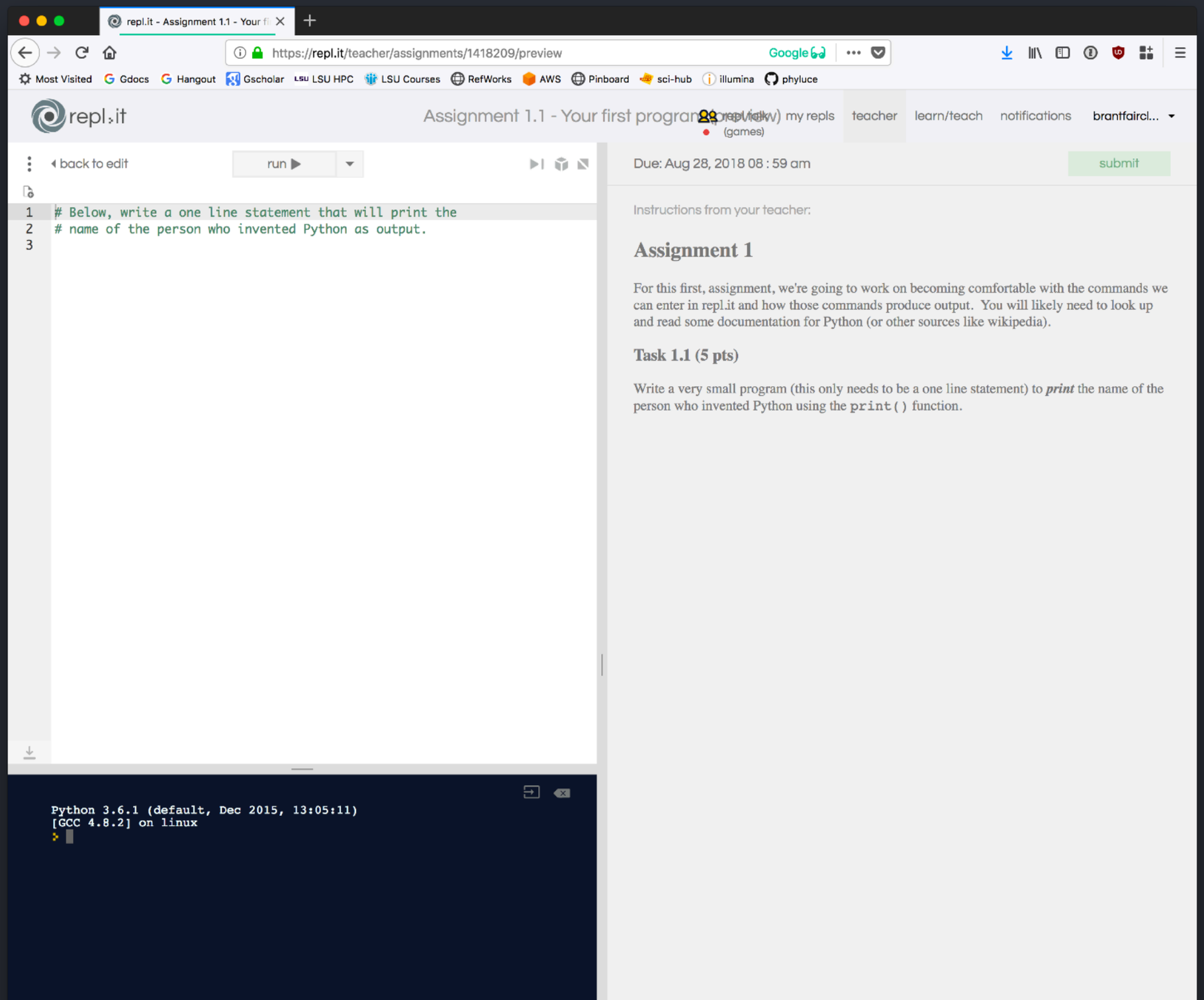
```
bcf at wasabi in ~
$ ipython
Python 2.7.11 |Anaconda 2.4.1 (x86_64)| (default, Dec  6 2015, 18:57:58)
Type "copyright", "credits" or "license" for more information.

IPython 4.0.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

Programs

How to run them



Programs

How to run them

hello_world.py

```python
#!/usr/bin/env python
# encoding: utf-8

"""
my first little program

Created by Brant Faircloth on 18 Jan 2016.
Copyright 2016 Brant C. Faircloth. All rights reserved.

"""

import sys

print("Python {}".format(sys.version_info))
print("Hello world")
```

2. zsh

```
bcf at wasabi in ~/Desktop
$ python hello_world.py
Python sys.version_info(major=2, minor=7, micro=11, releaselevel='final', serial=0)
Hello world
```

Programs

How to structure them

hello_world.py

```
#!/usr/bin/env python
# encoding: utf-8

"""
my first little program

Created by Brant Faircloth on 18 Jan 2016.
Copyright 2016 Brant C. Faircloth. All rights reserved.

"""

import sys

print("Python {}".format(sys.version_info))
print("Hello world")
```

"hash bang"

description

actual program

Programs

How to structure them

hello_world.py

```
#!/usr/bin/env python
# encoding: utf-8

"""
my first little program

Created by Brant Faircloth on 18 Jan 2016.
Copyright 2016 Brant C. Faircloth. All rights reserved.

"""

import sys


def main():
    print("Python {}".format(sys.version_info))
    print("Hello world")


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

function

"ifmain" stmnt

Programs

How **not** to structure them

You **do not** want **GIANT,** monolithic functions

```python
def combineLoci(self, record, min_distance):
    '''combined adjacent loci - this is somewhat cumbersome due to the
    format of the matches returned from msat (a dict with keys = motif).
    Essentially, we are running a pairwise comparison across all motifs
    located to determine which are within a predetermined distance from
    one another'''
    temp_combined = []
    reorder = ()
    # turn our dict into something more useful for this purpose
    for motif in record.matches:
        for pos,val in enumerate(record.matches[motif]):
            reorder += ((motif, pos, val[0][0], val[0][1], val[1], val[2]),)
    # sort it
    reorder = sorted(reorder, key=operator.itemgetter(2))
    # combine adjacent loci at < min_distance
    for i in reorder:
        included = False
        if not temp_combined:
            temp_combined.append([i])
        else:
            for gp, g in enumerate(temp_combined):
                for elem in g:
                    if i[2] - elem[3] <= min_distance:
                        temp_combined[gp].append(i)
                        included = True
                        break
            if not included:
                temp_combined.append([i])
    # re-key
    for key, group in enumerate(temp_combined):
        motifs = []
        if len(group) > 1:
            gs = group[0][2]
            ge = group[-1][3]
            gp = group[0][4]
            gf = group[-1][5]
        else:
            gs, ge = group[0][2], group[0][3]
            gp, gf = group[0][4], group[0][5]
        name = ''
        member_count = 0
        for pos,member in enumerate(group):
            if pos + 1 < len(group):
                dist = group[pos + 1][3] - group[pos][3]
                if dist > 1:
                    spacer = '...'
                else:
                    spacer = ''
            else:
                spacer = ''
            length = (member[3]-member[2])/len(member[0])
            name += '%s(%s)%s' % (member[0], length, spacer)
            motifs.append([member[0],length])
            member_count += 1
        record.combined[key] = (((gs, ge), gp, gf, member_count, motifs, name),)
    return record
```

Programs

How to structure them

You want **small**, **atomic** functions.

```python
import sys
import glob
import argparse
import ConfigParser
from phyluce import lastz
#from operator import itemgetter
from collections import defaultdict
import shutil

import pdb

class FullPaths(argparse.Action):
    """Expand user- and relative-paths"""
    def __call__(self, parser, namespace, values, option_string=None):
        setattr(namespace, self.dest, os.path.abspath(os.path.expanduser(values)))


class CreateDir(argparse.Action):
    def __call__(self, parser, namespace, values, option_string=None):
        # get the full path
        d = os.path.abspath(os.path.expanduser(values))
        # check to see if directory exists
        if os.path.exists(d):
            answer = raw_input("[WARNING] Output directory exists, REMOVE [Y/n]? ")
            if answer == "Y":
                shutil.rmtree(d)
            else:
                print "[QUIT]"
                sys.exit()
        # create the new directory
        os.makedirs(d)
        # return the full path
        setattr(namespace, self.dest, d)


def is_dir(dirname):
    if not os.path.isdir(dirname):
        msg = "{0} is not a directory".format(dirname)
        raise argparse.ArgumentTypeError(msg)
    else:
        return dirname


def is_file(filename):
    if not os.path.isfile:
        msg = "{0} is not a file".format(filename)
        raise argparse.ArgumentTypeError(msg)
    else:
        return filename


def get_name(header, splitchar = "_", items = 2):
    """use own function vs. import from match_contigs_to_probes - we don't want lowercase"""
    if splitchar:
        return "_".join(header.split(splitchar)[:items]).lstrip('>')
    else:
        return header.lstrip('>')
```