



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

Multiplication 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

Equivalence

"set variable to something"

"this is equal to that"

e.g. **x = 5**

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

```
27 def _mean_and_variance_pop_n(values):
28     n = 0
29     s = 0.0
30     ss = 0.0
31     for v in values:
32         n += 1
33         s += v
34         ss += v*v
35     if n == 0:
36         raise IndexError("values in mean_and_variance cannot be empty")
37     mean = float(s)/n
38     var = (ss - mean*s)/n
39     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)`

Out[3]: int

`type(2.6)`

Out[4]: float

`type("cat")`

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**, you can!

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

"brrrrrpris aonomatopoeia"

This is known as "string concatenation"

String Operations

But what about multiplication?

"brrrrrp" * 10

```
Out[6]: 'bbbbbbpbrrrrrpbrrrrrpbrrrrrpbrrrrrpbrrrrrpbrrrrrpbrrrrrpbrrrrrpbrrrrrpbrrrrr'
```

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)
```

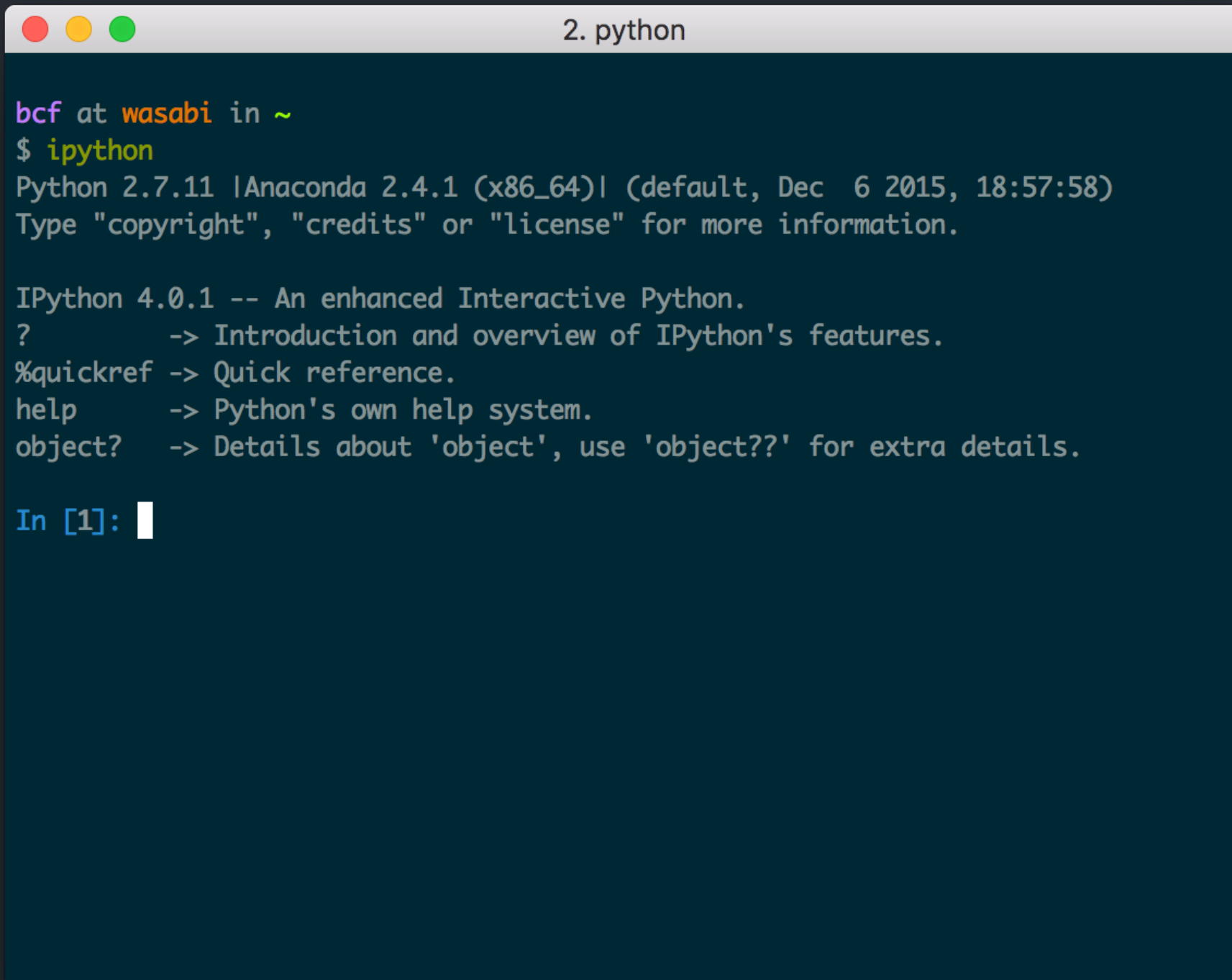
Debugging

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

The screenshot shows a web browser window with the URL `https://repl.it/teacher/assignments/1418209/preview`. The browser's address bar and tabs are visible at the top. The repl.it interface includes a header with the logo and navigation links like "teacher", "learn/teach", and "notifications". The main content area is titled "Assignment 1.1 - Your first program" and shows a code editor with the following Python code:

```
1 # Below, write a one line statement that will print the
2 # name of the person who invented Python as output.
3
```

Below the code editor is a terminal window displaying the Python environment information: "Python 3.6.1 (default, Dec 2015, 13:05:11) [GCC 4.8.2] on linux". The right sidebar contains instructions from the teacher, the assignment title "Assignment 1", a description of the task, and a "submit" button.

Due: Aug 28, 2018 08 : 59 am

Instructions from your teacher:

Assignment 1

For this first, assignment, we're going to work on becoming comfortable with the commands we can enter in repl.it and how those commands produce output. You will likely need to look up and read some documentation for Python (or other sources like wikipedia).

Task 1.1 (5 pts)

Write a very small program (this only needs to be a one line statement) to *print* the name of the person who invented Python using the `print ()` function.

Programs

How to run them

```
hello_world.py
1  #!/usr/bin/env python
2  # encoding: utf-8
3
4  """
5  my first little program
6
7  Created by Brant Faircloth on 18 Jan 2016.
8  Copyright 2016 Brant C. Faircloth. All rights reserved.
9
10 """
11
12 import sys
13
14 print("Python {}".format(sys.version_info))
15 print("Hello world")
16
```

```
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
1  #!/usr/bin/env python
2  # encoding: utf-8
3
4  """
5  my first little program
6
7  Created by Brant Faircloth on 18 Jan 2016.
8  Copyright 2016 Brant C. Faircloth. All rights reserved.
9
10 """
11
12 import sys
13
14 print("Python {}".format(sys.version_info))
15 print("Hello world")
16
```

"hash bang"

description

actual program

Programs

How to structure them

hello_world.py

```
1  #!/usr/bin/env python
2  # encoding: utf-8
3
4  """
5  my first little program
6
7  Created by Brant Faircloth on 18 Jan 2016.
8  Copyright 2016 Brant C. Faircloth. All rights reserved.
9
10 """
11
12 import sys
13
14
15 def main():
16     print("Python {}".format(sys.version_info))
17     print("Hello world")
18
19
20 if __name__ == '__main__':
21     main()
22
```

function

"ifmain" stmt

Programs

How **not** to structure them

You do not want
GIANT, monolithic
functions

```
1 def combineLoci(self, record, min_distance):
2     '''combined adjacent loci - this is somewhat cumbersome due to the
3     format of the matches returned from msat (a dict with keys = motif).
4     Essentially, we are running a pairwise comparison across all motifs
5     located to determine which are within a predetermined distance from
6     one another'''
7     temp_combined = []
8     reorder = ()
9     # turn our dict into something more useful for this purpose
10    for motif in record.matches:
11        for pos, val in enumerate(record.matches[motif]):
12            reorder += ((motif, pos, val[0][0], val[0][1], val[1], val[2]),)
13    # sort it
14    reorder = sorted(reorder, key=operator.itemgetter(2))
15    # combine adjacent loci at < min_distance
16    for i in reorder:
17        included = False
18        if not temp_combined:
19            temp_combined.append([i])
20        else:
21            for gp, g in enumerate(temp_combined):
22                for elem in g:
23                    if i[2] - elem[3] <= min_distance:
24                        temp_combined[gp].append(i)
25                        included = True
26                        break
27            if not included:
28                temp_combined.append([i])
29    # re-key
30    for key, group in enumerate(temp_combined):
31        motifs = []
32        if len(group) > 1:
33            gs = group[0][2]
34            ge = group[-1][3]
35            gp = group[0][4]
36            gf = group[-1][5]
37        else:
38            gs, ge = group[0][2], group[0][3]
39            gp, gf = group[0][4], group[0][5]
40        name = ''
41        member_count = 0
42        for pos, member in enumerate(group):
43            if pos + 1 < len(group):
44                dist = group[pos + 1][3] - group[pos][3]
45                if dist > 1:
46                    spacer = '...'
47                else:
48                    spacer = ''
49            else:
50                spacer = ''
51            length = (member[3] - member[2]) / len(member[0])
52            name += '%s(%s)%s' % (member[0], length, spacer)
53            motifs.append([member[0], length])
54            member_count += 1
55        record.combined[key] = ((gs, ge), gp, gf, member_count, motifs, name),)
56    return record
57
```


Programs

How to structure them

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

```
12 import sys
13 import glob
14 import argparse
15 import ConfigParser
16 from phyluce import lastz
17 #from operator import itemgetter
18 from collections import defaultdict
19 import shutil
20
21 import pdb
22
23 class FullPaths(argparse.Action):
24     """Expand user- and relative-paths"""
25     def __call__(self, parser, namespace, values, option_string=None):
26         setattr(namespace, self.dest, os.path.abspath(os.path.expanduser(values)))
27
28
29 class CreateDir(argparse.Action):
30     def __call__(self, parser, namespace, values, option_string=None):
31         # get the full path
32         d = os.path.abspath(os.path.expanduser(values))
33         # check to see if directory exists
34         if os.path.exists(d):
35             answer = raw_input("[WARNING] Output directory exists, REMOVE [Y/n]? ")
36             if answer == "Y":
37                 shutil.rmtree(d)
38             else:
39                 print "[QUIT]"
40                 sys.exit()
41         # create the new directory
42         os.makedirs(d)
43         # return the full path
44         setattr(namespace, self.dest, d)
45
46
47 def is_dir(dirname):
48     if not os.path.isdir(dirname):
49         msg = "{0} is not a directory".format(dirname)
50         raise argparse.ArgumentTypeError(msg)
51     else:
52         return dirname
53
54
55 def is_file(filename):
56     if not os.path.isfile(filename):
57         msg = "{0} is not a file".format(filename)
58         raise argparse.ArgumentTypeError(msg)
59     else:
60         return filename
61
62
63 def get_name(header, splitter = "_", items = 2):
64     """use own function vs. import from match_contigs_to_probes - we don't want lowercase"""
65     if splitter:
66         return "_".join(header.split(splitter)[:items]).rstrip('>')
67     else:
68         return header.rstrip('>')
```