Genome 373: Intro to Python I

Doug Fowler
Outline – Intro to Python I

• What is a program?
• Dealing with data
• Strings in Python
• Numbers in Python
What is a program?
What is a program?

A series of instructions, written in a language that is compiled or interpreted, intended to accomplish a task.
What is a program?

A series of instructions, written in a language that is compiled or interpreted, intended to accomplish a task.

```python
x = 4
y = 8
z = x + y
print(z)
```
What is a program?

A series of instructions, written in a language that is compiled or interpreted, intended to accomplish a task.

```
x = 4
y = 8
z = x + y
print(z)
```

Like a paragraph, programs are read from top to bottom, left to right.
What is a program?

A series of instructions, written in a language that is compiled or interpreted, intended to accomplish a task.

```
x = 4
y = 8
z = x + y
print(z)
```

Programming languages abstract machine language (different languages have different levels of abstraction).

Every programming language has specific syntax (e.g. grammar).
What is a program?

A series of instructions, written in a language that is compiled or interpreted, intended to **accomplish a task**.

\[
x = 4 \\
y = 8 \\
z = x + y \\
\text{print}(z)
\]
What is a program?

A series of instructions, written in a language that is compiled or interpreted, intended to **accomplish a task**

\[
\begin{align*}
x & = 4 \\
y & = 8 \\
z & = x + y \\
\text{print}(z)
\end{align*}
\]

Variables hold data; values can be reassigned
What is a program?

A series of instructions, written in a language that is compiled or interpreted, intended to **accomplish a task**

```python
x = 4
y = 8
z = x + y
print(z)
```

Functions perform an action on data.
What is a program?

A series of instructions, written in a language that is compiled or interpreted, intended to accomplish a task.

```python
x = 4
y = 8
z = x + y
print(z)
```

There are two other functions in this program – what are they?
What is a program?

A series of instructions, written in a language that is compiled or interpreted, intended to **accomplish a task**

```python
x = 4
y = 8
z = x + y
print(z)
```

There are two other functions in this program – what are they?

These satisfy our definition of a function (takes data; does something).

They are a special type of function called an operator

Operators are functions (typically basic/commonly used) that break syntax rules
What is a program?

A series of instructions, written in a language that is compiled or interpreted, intended to accomplish a task.

```python
x = 4
y = 8
z = x + y
print(z)
```

What is the output of this program?
What is a program?

A series of instructions, written in a language that is compiled or interpreted, intended to accomplish a task.

```
x = 4
y = 8
z = x + y
print(z)
```

What is the output of this program?

12
Data comes in different flavors

- True/False
- Boolean

ATGCTGCGC

string

42

number
Data comes in different flavors

True/False

Boolean

ATGCTGCGC

ATG, GCC, TAG

string

list

42

Gene1: 300, Gene2: 600, ...
GeneN: 150

number

hash
Why Python?

- **Python** is
  - easy to learn
  - fast enough
  - widely used
  - fairly portable

- **C** is much faster but much harder to learn and use.

- **Java** is somewhat faster but harder to learn and use.

- **Perl** is a little slower and a little harder to learn.
To follow along/try stuff out

- Login to the computer (just click)
- Open the Terminal either from the dock or from Applications -> Utilities
- Type “python” to launch the interactive interpreter
- If you have trouble, raise your hand and Matthew will come and help
- If you want get this slide deck to follow along, go to http://goo.gl/BPmlBu
Strings

• A **string** type object is a sequence of characters.
• In Python, string **literals** start and end with single or double quotes (but they have to match).

```python
>>> s = "foo"
>>> print s
foo
```
Strings

- A string type object is a sequence of characters.
- In Python, string literals start and end with single or double quotes (but they have to match).

```python
>>> s = "foo"
>>> print s
foo
>>> s = 'Foo'
>>> print s
Foo
>>> s = "foo"
SyntaxError: EOL while scanning string literal
```

(EOL means end-of-line; to the Python interpreter there was no closing double quote before the end of line)
Defining strings

- Each string is stored in computer memory as an array of characters in sequential bytes.

```python
>>> myString = "GATTACA"
```

The variable `myString` includes a pointer to the position in computer memory (the address) of the 0th byte above. Every byte in computer memory has a unique address.
Accessing single characters

• Access individual characters by using indices in square brackets.

```python
>>> myString = "GATTACA"
>>> myString[0]
'G'
>>> myString[2]
'T'
>>> myString[-1]
'A'
>>> myString[-2]
'C'
>>> myString[7]
Traceback (most recent call last):
  File "<stdin>" , line 1, in ?
IndexError: string index out of range
```
Accessing substrings ("slicing")

```python
>>> myString = "GATTACA"
>>> myString[1:3]
'AT'
>>> myString[:3]
'GAT'
>>> myString[4:]
'ACA'
>>> myString[3:5]
'TA'
>>> myString[:]
'GATTACA'
```

Notice that the length of the returned string [x:y] is y - x

Shorthand for beginning or end of string
Special characters

- The backslash is used to introduce a special character.

```python
>>> print "He said "Wow!"
SyntaxError: invalid syntax
```

```python
>>> print "He said \"Wow!\""
He said "Wow!"
```

```python
>>> print "He said:\nWow!"
He said:
Wow!
```

<table>
<thead>
<tr>
<th>Escape sequence</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\</td>
<td>Backslash</td>
</tr>
<tr>
<td>'</td>
<td>Single quote</td>
</tr>
<tr>
<td>&quot;</td>
<td>Double quote</td>
</tr>
<tr>
<td>\n</td>
<td>Newline</td>
</tr>
<tr>
<td>\t</td>
<td>Tab</td>
</tr>
</tbody>
</table>

whenever Python runs into a backslash in a string it interprets the next character specially
More string functionality

```python
>>> len("GATTACA")
7
>>> print "GAT" + "TACA"
GATTACA
>>> print "A" * 10
AAAAAAAAAAA
>>> "GAT" in "GATTACA"
True
>>> "AGT" in "GATTACA"
False
>>> temp = "GATTACA"
>>> temp2 = temp[1:4]
>>> print temp2
ATT
>>> print temp
GATTACA
```
String methods

- In Python, a **method** is a function that is defined for a particular type of object.

- The syntax is:
  
  ```python
  object.method(arguments)
  ```

  or **object.method()** - no arguments

```python
>>> dna = "ACGT"
>>> dna.find("T")
3
```

- The first position where "T" appears

**object (in this case a string object)**

**string method**

**method argument**
Some of many string methods

>>> s = "GATTACA"
>>> s.find("ATT")
1
>>> s.count("T")
2
>>> s.lower()
'gattaca'
>>> s.upper()
'GATTACA'
>>> s.replace("G", "U")
'UATTACA'
>>> s.replace("C", "U")
'GATTAUA'
>>> s.replace("AT", "**")
'G**TACA'
>>> s.startswith("G")
True
>>> s.startswith("g")
False
Strings are immutable

- Strings cannot be modified; instead, create a new string from the old one using assignment.

```python
>>> s = "GATTACA"
>>> s[0] = "R"
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: 'str' object doesn't support item assignment
>>> s = "R" + s[1:]
>>> print s
RATTACA
>>> s = s.replace("T","B")
>>> print s
RABBACA
>>> s = s.replace("ACA", "I")
>>> print s
RABBI
>>> s
'RABBI'
```
Strings are immutable

- String methods do not modify the string; they return a new string.

```python
>>> seq = "ACGT"
>>> seq.replace("A", "G")
'GCGT'
>>> print seq
ACGT
>>> new_seq = seq.replace("A", "G")
>>> print new_seq
GCGT
>>> print seq
ACGT
```
Basic string operations:
S = "AATTGG"
# literal assignment - or use single quotes ''
s1 + s2
# concatenate
S * 3
# repeat string
S[i]
# get character at position 'i'
S[x:y]
# get a substring
len(S)
# get length of string
int(S)
# turn a string into an integer
float(S)
# turn a string into a floating point decimal number

Methods:
S.upper()
S.lower()
S.count(substring)
S.replace(old,new)
S.find(substring)
S.startswith(substring)
S.endswith(substring)

Printing:
print var1,var2,var3
# print multiple variables
print "text",var1,"text"
# print a combination of literal text (strings) and variables

# is a special character – everything after it is a comment, which the program will ignore – USE LIBERALLY!!
Numbers

- Python defines various types of numbers:
  - Integer (1234)
  - Floating point number (12.34)
  - Octal and hexadecimal number (0177, 0x9gff)
  - Complex number (3.0+4.1j)

- You will likely only use the first two.
Conversions

>>> 6/2
3
>>> 3.0/4.0
0.75
>>> 3/4.0
0.75
>>> 3*4.0
12.0
>>> 3*4
12
>>> 3/4
0

• The result of a mathematical operation on two numbers of the same type is a number of that type.

• The result of an operation on two numbers of different types is a number of the more complex type.

• The result of integer divisions are truncated rather than rounded.
Formatting numbers

• The % operator formats a number.
• The syntax is `<format> % <number>`
• `format` is a string

```python
>>> "%f" % 3   # print as float
'3.000000'
>>> "%.2f" % 3 # print as float with
'3.00'       # 2 digits after decimal
>>> "%5.2f" % 3 # width 5 characters
'3.00'
```
Formatting codes

- `%i` = integer (or `%d`)
- `%f` = float value (decimal number)
- `%e` = scientific notation
- `%g` = general, easily readable notation (uses decimal notation unless there are too many zeroes, then switches to scientific notation)
More complex formats

%[flags][width][.precision][code]

- Left justify ("-")
- Include numeric sign ("+")
- Fill in with zeroes ("0")

Total width of output

Number of digits after decimal

i, f, e, g
Examples (review later)

```python
>>> x = 7718
>>> "%i" % x
'7718'
>>> "%-6i" % x
'7718  '
>>> "%06i" % x
'007718'
>>> x = 1.23456789
>>> "%i" % x
'1'
>>> "%f" % x
'1.234568'
>>> "%e" % x
'1.234568e+00'
>>> "%g" % x
'1.23457'
>>> "%g" % (x * 10000000)
'1.23457e+07'
```

Don’t worry if this all looks like Greek – you can figure out how to do these when you need them in your programs. After a while they are pretty easy.

It σουρε λοοκσ λικε Γρεεκ το με.
(It sure looks like to Greek to me)
Sample problem #1

- Write a program called dna2rna.py that reads a DNA sequence from the first command line argument and prints it as an RNA sequence. Make sure it retains the case of the input.

```
> python dna2rna.py  ACTCAGT
ACUCAGU

> python dna2rna.py  actcagt
acucagu

> python dna2rna.py  ACTCagt
ACUCAgu
```

Hint: first get it working for uppercase letters and then extend it to lowercase and mixed case.
Two solutions

```python
import sys
seq = sys.argv[1]
new_seq = seq.replace("T", "U")
newer_seq = new_seq.replace("t", "u")
print newer_seq

OR

import sys
print sys.argv[1]  (to be continued)
```
Two solutions

```python
import sys
seq = sys.argv[1]
new_seq = seq.replace("T", "U")
newer_seq = new_seq.replace("t", "u")
print newer_seq

import sys
print sys.argv[1].replace("T", "U")  # (to be continued)
```
Two solutions

import sys
seq = sys.argv[1]
new_seq = seq.replace("T", "U")
newer_seq = new_seq.replace("t", "u")
print newer_seq

import sys
print sys.argv[1].replace("T", "U").replace("t", "u")

• It is legal (but not always desirable) to chain together multiple methods on a single line.
• Think through what the second program does, going left to right, until you understand why it works.
Sample problem #2

- Write a program `get-codons.py` that reads the first command line argument as a DNA sequence and prints the first three codons, one per line, in uppercase letters.

```bash
> python get-codons.py TTGCAGTCG
TTG
CAG
TCG
> python get-codons.py TTGCAGTCGATCTGATC
TTG
CAG
TCG
> python get-codons.py tcgatcgactg
TCG
ATC
GAC
```

(slight challenge - print the codons on one line separated by spaces)
# program to print the first 3 codons from a DNA sequence given as the first command-line argument

```python
import sys
seq = sys.argv[1]  # get first argument
up_seq = seq.upper()  # convert to upper case
print up_seq[0:3]  # print first 3 characters
print up_seq[3:6]  # print next 3
print up_seq[6:9]  # print next 3
```

These comments are simple, but when you write more complex programs good comments will make a huge difference in making your code understandable (both to you and others).
Sample problem #3

• Write a program that reads a protein sequence as a command line argument and prints the location of the first cysteine residue (C).

```bash
> python find-cysteine.py
MNDLSGKTVIITGGARGLGAEAARQAVAAGARVVLADVLDEEGAATARELGDAARYQHLDVTI
EEDWQRVCAYAREEFGSVDGL
70
> python find-cysteine.py
MNDLSGKTVIITGGARGLGAEAARQAVAAGARVVLADVLDEEGAATARELGDAARYQHLDVTI
EEDWQRVVAYAREEFGSVDGL
-1
```

note: the −1 here means that no C residue was found
import sys
protein = sys.argv[1]
upper_protein = protein.upper()
print upper_protein.find("C")

(Always be aware of upper and lower case for sequences - it is valid to write them in either case. This is handled above by converting to uppercase so that 'C' and 'c' will both match.)
Challenge problem

- Write a program `get-codons2.py` that reads the first command-line argument as a DNA sequence and the second argument as the frame, then prints the first three codons in that frame on one line separated by spaces.

```bash
> python get-codons2.py TTGCAGTCGAG 0
TTG  CAG  TCG
> python get-codons2.py TTGCAGTCGAG 1
TGC  AGT  CGA
> python get-codons2.py TTGCAGTCGAG 2
GCA  GTC  GAG
```
import sys
seq = sys.argv[1]
frame = int(sys.argv[2])
seq = seq.upper()
c1 = seq[frame:frame+3]
c2 = seq[frame+3:frame+6]
c2 = seq[frame+6:frame+9]
print c1, c2, c3
Python review (from quiz section)

• interactive and script mode
  - use interactive mode to check syntax etc.
  - mostly use script mode: write a program, run it

• python code is built with **objects**, each of which has a **type**

• a **variable** is a name for an object

• the assignment operator is `'='`
  e.g. `myString = "spring is here!"

• the **import** command adds functionality not available by default

• command-line arguments are accessed through **sys.argv**