Python I: Printing and Manipulating Text

The set up

Log into newton head node, get an interactive session, and get python loaded

```bash
$ ssh username@newlogin.newton.utk.edu
$ qrsh xterm
$ hostname
$ module load python
```

The hostname command should return the name of a newton computer (a latin name like chi or psi followed by a number).

Check the python version with the flag -V

```bash
$ python -V
```

Printing

We are going to start learning python through working with text. Most programmers need numbers first, so that is a traditional approach. However, biologists often need to work with text - DNA or protein sequences in particular.

**string**

"String" is a word used by computer scientists to describe some text.
printing a message to the screen

The traditional first program when learning a new computer language is to print "hello, world" to the screen. We wouldn't want to ignore tradition, so let's do this first.

Let's start a new directory for holding all our python materials.

```
$ mkdir learning_python
$ cd learning_python
```

Now use your favorite text editor to open a file called hello.py. In that file put:

```
print ("Hello world")
```

function

A function performs some task, in this case printing things. A function is followed by parenthesis (usually; best to assume you need them)

arguments

In the parentheses after a function, you can put arguments. Arguments provide information to the function, in this case what to print.

Run the program.

```
$ python hello.py
```

Just like in shell scripting, we can comment the code to provide clarity. In a new python script named comment.py, place the following:

```
# this is a comment, it will be ignored by the computer
print("Comments are very useful!")
```
and test it out:

```
$ python comment.py
```

## Exploring errors

Computer programs almost never work correctly the first time. Programming languages are not like natural languages – they have a very strict set of rules, and if you break any of them, the computer will not attempt to guess what you intended, but instead will stop running and present you with an error message. You're going to be seeing a lot of these error messages in your programming career, so let's get used to them as soon as possible.

Let's explore some errors that could occur in our hello, world program.

- Remove the quotes (Yields: SyntaxError)
- Misspell print as prin (Yields: NameError)
- Splitting a statement over two lines (Yields: SyntaxError)

## Printing special characters

We need is a way to include a new line as part of a string, and luckily for us, Python has just such a tool built in. To include a new line, we write a backslash followed by the letter n – Python knows that this is a special character and will interpret it accordingly. Here's the code which prints "Hello world" across two lines:

```python
# how to include a new line in the middle of a string
print("Hello\nworld")
```

Notice that there's no need for a space before or after the new line. Another useful character is a tab (\t).
Storing strings in variables

We can take a string and **assign a name to it using an equals sign** – we call this a variable.

```python
# store a short DNA sequence in the variable my_dna
my_dna = "ATGCGTA"
```

The variable `my_dna` now points to the string "ATGCGTA". We call this assigning a variable, and once we've done it, we can use the variable name instead of the string itself – for example, we can use it in a print statement:

```python
# store a short DNA sequence in the variable my_dna
my_dna = "ATGCGTA"

# now print the DNA sequence
print(my_dna)
```

Notice that here I have used a blank line to separate out the two bits of the program (calculating the counts, and printing them). Python is perfectly happy with this – it just ignores blank lines, so it’s fine to put them in in order to make your programs more readable for humans.

The equals operator will allow you to assign the same variable to a new value.

```python
# store a short DNA sequence in the variable my_dna
my_dna = "ATGCGTA"
my_dna = "CGTTTA"

# now print the DNA sequence
print(my_dna)
```

Variable names are arbitrary – that means that we can pick whatever we like to be the name of a variable. So our code above would work
We like to be the name of a variable. So our code above would work in exactly the same way if we picked a different variable name:

```python
# store a short DNA sequence in the variable banana
banana = "ATGCAGT"
# now print the DNA sequence
print(banana)
```

What makes a good variable name? Generally, it's a good idea to use a variable name that gives us a clue as to what the variable refers to. In this example, `my_dna` is a good variable name, because it tells us that the content of the variable is a DNA sequence. Conversely, `banana` is a bad variable name, because it doesn't really tell us anything about the value that's stored.

There are some rules about selecting variable names:

- use letters, numbers, and underscores only (no odd characters like £, ^ or %)
- cannot start with a number (though numbers can be in the middle or at the end of a name)
- variable names are case sensitive, so `mydna`, `MyDNA`, and `MY_DNA` are all different
- can't use a word that's already built in to the Python language like "print".

## Tools for manipulating strings

### Concatenation

We can concatenate (stick together) two strings using the + symbol. This symbol will join together the string on the left with the string on the right:

```python
my_dna = "AATT" + "GGCC"
print(my_dna)
```
In the above example, the things being concatenated were strings, but we can also use variables that point to strings:

```python
upstream = "AAA"
my_dna = upstream + "ATGC"
# my_dna is now "AAAATGC"
```

It's important to realize that the result of concatenating two strings together is itself a string. So it's perfectly OK to use a concatenation inside a print statement:

```python
print("Hello" + " " + "world")
```

Using one tool inside another is quite a common thing to do in Python.

**Length**

**len**

Another useful built-in tool in Python is the `len` function (len is short for length). Just like the print function, the len function takes a single argument (take a quick look back at when we were discussing the print function for a reminder about what arguments are) which is a string. However, the behaviour of the len function is quite different. Instead of outputting text to the screen, len outputs a value that can be stored – we call this the return value. In other words, if we write a program that uses len to calculate the length of a string, the program will run but we won't see any output:

```python
# this line doesn't produce any output
len("ATGC")
```

If we want to actually use the return value, we need to store it in a variable, and then do something useful with it (like printing it):
There's another interesting thing about the len function: the result (or return value) is not a string, it's a number. This is a very important idea: **Python treats strings and numbers differently.**

We can see that this is the case if we try to concatenate together a number and a string. Consider this short program which calculates the length of a DNA sequence and then prints a message telling us the length:

```python
# store the DNA sequence in a variable
my_dna = "ATGCGAGT"
# calculate the length of the sequence and store it in a variable
dna_length = len(my_dna)
# print a message telling us the DNA sequence length
print("The length of the DNA sequence is " + dna_length)
```

When we try to run this program, we get the following error: `TypeError: cannot concatenate 'str' and 'int' objects`. Python is complaining that it doesn't know how to concatenate a string (which it calls str for short) and a number (which it calls int – short for integer). Strings and numbers are examples of types – different kinds of information that can exist inside a program.

**str**

Python has a built-in solution – a function called `str` which turns a number into a string so that we can print it. Here's how we can modify our program to use it:

```python
# store the DNA sequence in a variable
```
# Store the DNA sequence in a variable
my_dna = "ATGCGAGT"

# calculate the length of the sequence and store it in a variable
dna_length = len(my_dna)

# print a message telling us the DNA sequence length
print("The length of the DNA sequence is " + str(dna_length))

**Substrings**

What do we do if we have a long string, but we only want a short portion of it? This is known as taking a substring, and it has its own notation in Python. To get a substring, we follow the variable name with a pair of square brackets which enclose a start and stop position, separated by a colon.

However, the start and stop position behavior is not intuitive. First, Python starts counting at 0. So a five letter word, such as "amino," the indices for the letters are 0, 1, 2, 3, 4. Second, the range positions are inclusive at the start, but exclusive at the stop. In other words, the expression protein[3:5] gives us everything starting at the third character, and stopping just before the fifth character (i.e. characters three and four).

Let's use a protein sequence to demonstrate:

```python
protein = "vlspadktnv"

# print positions three to five
print(protein[3:5])

# positions start at zero, not one
print(protein[0:6])
```
counting substrings

A very common job in biology is to count the number of times some pattern occurs in a DNA or protein sequence. In computer programming terms, what that translates to is counting the number of times a substring occurs in a string. The method that does the job is called count. It takes a single argument whose type is string, and returns the number of times that the argument is found in the variable. The return type is a number, so be careful about how you use it!

Remember that we have to use str to turn the counts into strings so that we can print them.

protein = "vlspadktnv"

# count amino acid residues
valine_count = protein.count('v')
lsp_count = protein.count('lsp')
tryptophan_count = protein.count('w')

# now print the counts
print("valines: " + str(valine_count))
print("lsp: " + str(lsp_count))
print("tryptophans: " + str(tryptophan_count))

finding substrings

What if instead of counting the number of proline residues in our protein sequence we want to know where they are? The find method will give us the answer, at least for simple cases. find takes a single string argument, just like count, and returns a number which is the position at which that substring first appears in the string (in
position at which that substring first appears in the string (in computing, we call that the index of the substring).

Remember that in Python we start counting from zero rather than one, so position 0 is the first character, position 4 is the fifth character, etc. A couple of examples:

```python
protein = "vlspadktnv"
print(str(protein.find('p')))
print(str(protein.find('kt')))
print(str(protein.find('w')))
```

Notice the behaviour of find when we ask it to locate a substring that doesn't exist – we get back the answer -1.

Both count and find have a pretty serious limitation: you can only search for exact substrings. If you need to count the number of occurrences of a variable protein motif, or find the position of a variable transcription factor binding site, they will not help you. (See Chapter 7 of Python for Biologists for the life-changing magic of regular expressions).

**Attribution**

This material is made available under the Creative Commons Attribution license. Much of the material has been adapted from the book Python for Biologists by Martin Jones. If you found this material useful, please consider buying the book! [http://pythonforbiologists.com]