

Python 101

Input and Output

DIOGO SILVA

Input and output channels

There are three input/output channels that allow the program to interact with the environments and its users:

- Standard input: From where the scripts reads the input data (default: computer keyboard)
- Standard output: Where the scripts directs the output data (default: computer terminal screen)
- Standard error: Where the scripts directs error messages during the execution of the script (default: computer terminal screen)

Input from user/keyboard

Python has two built-in functions for reading data provided by the user via the keyboard:

- **`raw_input([prompt])`**: this function reads one line from the standard input and returns it as a string

```
sequence = raw_input("Please provide a DNA sequence:\n>")
print sequence
```

- **`input([prompt])`**: this function is similar to `raw_input()`, except that it assumes the input is a valid python expression and returns the evaluated result to you. It can interpret if you are providing a string or a number, by using quotations marks or not.

```
sequence = input ("Please provide a DNA sequence:\n>")
n_loci = input ("Please provide the number of loci:\n>")
```

Printing the output on the terminal

- The *print* keyword can be used to print any type of objects into the computer terminal screen.

```
1 Sample = ["H.sapiens", "C.lupus", "M.musculus"]
2 for species in Sample: print species
```

01/10/2014 10:42 AM

```
H.sapiens  
C.lupus  
M.musculus
```

- Note that print adds a *newline* (\n) character at the end of the line. To avoid this, a comma can be put after the object that you want to print.

```
1 Sample = ["H.sapiens","C.lupus","M.musculus"]  
2 for species in Sample: print species,  
3
```

```
H.sapiens C.lupus M.musculus
```

Dealing with files

Open and Create file objects

- **Open()** returns a file object and may take two arguments: *open(filename,mode)*, where mode can be "r" (read), "w" (write), "rw" (both read and write) or "a" (append)

```
1 read_file = open("my_file.fas","r")  
2 new_file = open("my_new_file.fas","w")  
3 append_file = open("my_file_append.fas","a")  
5
```

Pay special attention that:

- In modes "w" and "a", if the specified filename does not exist, the *open()* function creates it automatically.
 - In mode "w", if an existent file is specified in the *filename*, the original file is overwritten.
 - In mode "a", any data written to the file is automatically added to the end.
-

Dealing with files

Methods for reading file objects

```
1 read_file = open("my_file.fas","r")
2 read_file.read() # reads the whole file and returns the
  content in one string
3 read_file.read([N]) # reads the file up to N bytes and
  returns a string
4 read_file.readline() # reads a single line of the file and
  returns a string
5 read_file.readlines() # reads all lines and returns a list
7 of lines
```

- Note that all these methods exhaust the file contents, but these can still be assigned to variables and used multiple times

```
1 content = read_file.readlines() # Consumes lots of RAM
  memory for large files
2 red_file.readlines()
3 print content
5
```

Dealing with files

Methods for reading file objects

- Alternatively, the contents of a file can be read with a *for* loop in a line-by-line basis.

```
1 read_data = open("my_file.fas")
2 for line in read_data:
3     print line
5
```

Python. This is also much faster and is memory efficient than assigning the whole content of a file to a variable because only one line is actually stored in memory in each loop iteration

Dealing with files

Writing to files

- To write data on a file, the `write()` method can be used

```
1 output_file = open("New_file.fas", "w")
2 output_file.write("Hello world!\nI am writing in a new
3 line!\n\tAnd now it's indented!")
4
```

- Using this method, you can only write *string* objects into a file. If you wish to write something other than a string, it needs to be converted to a string first

```
1 data = "The taxa", ["H.sapiens", "C.lupus"], "have",
2 (2,2), "stop and start codons, respectively"
3 data_str = str(data)
4
```

Closing files

A file is automatically closed when the program ends. However, if you are done with a file, you can close it and free up any system resources with the `close()` method

```
1 output_file.close()
2 output_file.read() # The file object no longer exists
3
4
```