

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

```
1 sequence = raw_input("Please provide a DNA
2 sequence:n>")
4 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.

```
1 sequence = input ("Please provide a DNA sequence:n>")
2 n_loci = input ("Please provide the number of
4 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  #!/usr/bin/python
2  Sample = ["H.sapiens", "C.lupus", "M.musculus"]
3  for species in Sample: print species
4
```

```
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  #!/usr/bin/python
2  Sample = ["H.sapiens", "C.lupus", "M.musculus"]
3  for species in Sample: print species,
4
```

```
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  #!/usr/bin/python
2  read_file = open("my_file.fas", "r")
3  new_file = open("my_new_file.fas", "w")
4  append_file = open("my_file_append.fas", "a")
6
```

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

- File objects can be read using several built-in methods

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

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

```
1  #!/usr/bin/python
2  content = read_file.readlines() # Consumes lots of RAM
3  memory for large files
4  read_file.readlines()
6  print content
```

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  #!/usr/bin/python
2  read_data = open("my_file.fas")
3  for line in read_data:
4      print line
6
```

This is also much faster and 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  #!/usr/bin/python
2  output_file = open("New_file.fas","w")
3  output_file.write("Hello world!nI am writing in a new
5  line!ntAnd now it's indented!")
```

- 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  #!/usr/bin/python
2  data = "The taxa",["H.sapiens","C.lupus"],"have",
3  (2,2),"stop and start codons, respectively"
5  data_str = str(data)
```

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  #!/usr/bin/python
2  output_file.close()
3  output_file.read() # The file object no longer exists
5
```