Python Scripting 1

657.019 Scripting for Biotechnologists (WS 2018/19)

Why Python?

- Most used scripting language in bioinformatics
- Desirable skill to have
  - Along with R, Perl and shell scripting
- Python vs. Perl
  - Python is easier to learn
    - Enforced coding style leads to better readability
  - Perl is powerful but less popular (steep learning curve)
- Goal
  - To learn basics of Python scripting (we use Python 3)

Objectives

- Part 1: To learn how to
  - Work with text
  - Work with files
  - Work with lists
  - Use loops
- Part 2: To learn how to
  - Write my own functions
  - Use conditionals
  - Work with dictionaries
- Part 3: To learn how to
  - Interact with operating systems
  - Use basic Biopython

Running Python scripts

- Interactive interpreter
  - Type **`python`** at shell prompt
  - Enter statement at Python prompt (**>>>**), hit Enter for result

- Run script file
  - Create a file starting with **#!** : **`myscript.py`**

```
#!/usr/local/bin/python3
number = 3
print(number)
```

  - Run it with either command

    `python3 myscript.py`

    `./myscript.py` (if **`myscript.py`** is set to be executable)

Basic text manipulation

- Printing

```
print("Hello world")
print("Hello\nworld")
```

- Storing in a variable (assignment)

```
my_seq = "ACTGTTA"
print(my_seq)
```

- Concatenation

```
my_seq = "GCGC" + "TATA"
print(my_seq)                    # GCGCTATA
```

- Getting length

```
my_seq = "ACTGTTA"
seq_len = len(my_seq)            # 7 (as number)
print("Sequence has " + seq_len + " bases")       # fails
print("Sequence has " + str(seq_len) + " bases") # works
```

Advanced text manipulation

- Case conversion

```
my_seq = "ACTGTTA"
print(my_seq.lower()) # actgtta
```

  - `upper()` also available for strings

- Replacement

```
my_seq = "ACTGTTA"
print(my_seq.replace("T", "U")) # ACUGUUA
print(my_seq)                   # ACTGTTA
```

- Extracting substring

```
my_seq = "ACTGTTA"
print(my_seq[2:5]) # TGT (0-based, excluding stop)
print(my_seq[2:])  # TGTTA
```

More string methods

- Counting

```
my_seq = "ACTGTTA"
A_count = my_seq.count("A")   # 2
N_count = my_seq.count("N")   # 0
```

- Finding position (index)

```
my_seq = "ACTGTTA"
print(str(my_seq.find("C"))   # 1
print(str(my_seq.find("TT")) # 4
print(str(my_seq.find("t"))   # -1: not found
```

- Methods
  - Defined for a particular type (e.g. string) or types
  - Can (but not always) change the object that they operate on

Reading from a file

- File: `seq.txt`

```
TAAACGACCTAGATC
GCTAGGCATAA
```

- Opening file, reading lines

```
my_file = open("seq.txt")     # create file object
line1 = my_file.readline()    # TAAACGACCTAGATC\n
line2 = my_file.readline()    # GCTAGGCATAA\n
line2 = line2.rstrip()        # GCTAGGCATAA
```

Writing to a file

- Files can be open in three modes
  - Read ("r", default), write ("w"), append ("a")

- Opening file, writing to it

```
my_file = open("info.txt", "w")
my_file.write("acgt" + "nnnn")      # acgtnnnn
my_file.write(str(len("TTAATT"))) # 6
my_file.write("GCTAA".lower())      # gctaa
```

- Closing file

```
my_file.close()
```

Lists

- A list contains ordered things (objects)

- List manipulation

```
ranks=["do","ph","cl","or","fa","ge"]
len(ranks)                  # 6
high_ranks = ranks[0:3]# do,ph,cl
# Extract up to 1 before ending index
ranks.append("sp")          # do,ph,cl,or,fa,ge,sp
ranks.reverse()             # sp,ge,fa,or,cl,ph,do
ranks.sort()                # cl,do,fa,ge,or,ph,sp
ranks2 = ["spp"]
ranks + ranks2              # cl,do,fa,ge,or,ph,sp,spp
ranks.extend(ranks2)        # cl,do,fa,ge,or,ph,sp,spp
```

- List methods can change the list itself

Looping over list

- Using literal list

```
genes=["APP","GAST","INS","LCK","PIP"]
for gene in genes:
  print(gene + " is a human gene")
  print("with name starting with " + gene[0])
```

- Using string as list

```
seq = "ACGTGGCA"
for base in seq:                # prints A, C, G, …
  print("base: " + base)
```

- Splitting string into list

```
seq = "ACG,TCG,CCT"
triplets = seq.split(",")    # ['ACG', 'TCG', 'CCT']
for triplet in triplets:
  print("triplet: " + triplet)
```

Indentation rules

- A code block is indicated by indentation

```
for gene in genes:      # : required before block begins
  print(gene + " is a human gene")              # 2 spaces
  print("with name starting with " + gene[0])# 2 spaces
```

- Block scopes indicated by relative amounts of indentation
  - Code at the same level must have same number of leading spaces
    - Number of spaces doesn't matter (1, 2, 4, …)
  - Inner block must have one or more leading spaces than immediately surrounding block
- When rules are violated:
  - **`IndentationError`**

Looping over lines of a file

- Do something for each line of a file, be done

```
file = open("dna.txt")
for line in file:          # end of file after for loop
  print("A count: " + str(line.count("A")))
```

- Save lines of a file, do more than one thing

```
file = open("dna.txt")
lines = file.readlines() # lines form a list
for line in lines:
  print("A count: " + str(line.count("A")))
print(lines[0:2])
```

Looping over range

- Generating number ranges

```
r = range(5)          # 0,1,2,3,4
r = range(0,10)       # 0,1,2,3,4,5,6,7,8,9
r = range(0,10,2)     # 0,2,4,6,8
r = range(10,0,-2)    # 10,8,6,4,2
r[2]                  # 6
```

- Looping over range

```
for num in range(5):  # prints A0,A1,A2,A3,A4
  print("A" + str(num))
```