Python Scripting 2

657.019 Scripting for Biotechnologists (WS 2018/19)

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

How to use a function

- First define it

```
def gc_ratio(seq):
    seq_len = len(seq)
    g_count = seq.count("G")
    c_count = seq.count("C")
    gc_ratio = (g_count + c_count)/seq_len
    return gc_ratio
```

- Then call it

```
seq = "ACGTACGTACGT"
print("Seq:", seq, " GC ratio:", gc_ratio(seq))
# Seq: ACGTACGTACGT  GC ratio: 0.5
```

- Indentation matters!

Programming a function

- Build a quick, imperfect prototype, then improve
- Test our GC ratio function
  1. `gc_ratio("acgt")` fails: why?
  2. `gc_ratio("GGCCGGCATAT")` works, but: `0.6363636363636…`
  3. `gc_ratio("ACGTNN")` works, but the ratio is not accurate
- Solutions
  1. Convert sequence characters to uppercase
     - Already learned a string method to do this
  2. Round the ratio
     - Find a function to do this
  3. Remove N bases before getting sequence length
     - Another string method can be used

Multi-argument functions

- Improved GC ratio function

```
def gc_ratio(seq, sig_digits):
    # convert to uppercase, remove N's
    seq = seq.upper().replace("N", "")
    seq_len = len(seq)
    g_count = seq.count("G")
    c_count = seq.count("C")
    gc_ratio = (g_count + c_count)/seq_len
    return round(gc_ratio, sig_digits)
    # round(0.636363636, 2): 0.64
```

Kinds of functions

- No argument

```
def hello():
    print("Hello world")
```

- No return value

```
def print_error(message):
    print("Error:", message)
```

- Default argument values

```
def gc_ratio(dna, sig_digits=2):
```

  - Can be called as:
    - `gc_ratio("acgtACGT")`
    - `gc_ratio("acgtACGT", 4)`
    - `gc_ratio("acgtACGT", sig_digits=3)`
    - `gc_ratio(dna="acgtACGT", sig_digits=3)`
    - `gc_ratio(sig_digits=3, dna="acgtACGT")`

Testing for conditions

- Tests return either **`True`** or **`False`**
  - `7 == 8` `# False`
  - `7 < 8` `# True`
  - `7 >= 9` `# False`
  - `7 != 10` `# True`
  - `"ATGC".startswith("AT")` `# True`
  - `"ATGC".endswith("A")` `# False`
  - `"ATGC".endswith("")` `# ???`
  - `"TGgT".isupper()` `# False`
  - `"N" in ["A","C","G","T"]` `# False`
  - `"A" is "A"` `# True`

- Tests can be combined with **`and`**, **`or`**, and **`not`**
- **`True`** and **`False`** are built-in values, not strings

Conditionals

- **If-elif-else** structure is all you need

```
if hit_count > 100:
  print("High hits")
elif hit_count > 50:
  print("Medium hits")
else:
  print("Low hits")
```

- Combined tests

```
genes=["APP","GAST","INS","LCK","GIP"]
for gene in genes:
  if gene.startswith("G") and gene.endswith("T"):
    print(gene)
```

True/False functions

- Functions returning yes/no are useful

```
def is_gc_ratio_big(seq):
    seq = seq.upper().replace("N", "")
    seq_len = len(seq)
    g_count = seq.count("G")
    c_count = seq.count("C")
    gc_ratio = (g_count + c_count)/seq_len
    if (gc_ratio > 0.7):    # Much simpler is:
        return True         # return gc_ratio > 0.7
    else:
        return False
```

Dictionaries (hashes)

- Key -> value pairs of data

```
taxid2species = {9796:"Equus caballus", 9685:"Felis catus", 9612:"Canis lupus"}
taxid2species = {}
taxid2species[9796] = "Equus caballus" # and so on
```

- Getting value

```
taxid2species.get(9685)     # Felis catus
taxid2species.get(9606, 0) # 0 (default if key not found)
```

- Removing key

```
taxid2species.pop(9685)
```

- Getting keys

```
taxid2species.keys()     # [9796, 9612]
```

Dictionary example

- Count frequencies of di-nucleotides

```
seq = "TCGTAAACATCAAAAGT"
dinucls = ['AA','AT','AG','AC','TA','TT','TG',
'TC','GA','GT','GG','GC','CA','CT','CG','CC']
di2count = {}
for di in dinucls:
  count = seq.count(di)
  if count > 0:
    di2count[di] = count:
print(di2count)          # {'TA':1,'AA':3, 'AT':1, …}
```

Iteration over keys

- Print di-nucleotides occurring just once

```
for di in di2count.keys():
  if di2count.get(di)==1:
    print(di)
```

- Sorting keys before iteration
  - Keys are returned in random order

```
for di in sorted(di2count.keys()):
```