Python Cheatsheet *for BMI3*

Yuchen SHEN

Python Basics

IO

Deal with given json-encoded data:

```
input_data= json.load(sys.stdin)
dnas= input_data["dnas"]
k= input_data["k"]
```

Arithmetic operations

```
10 * 3 # 30
10 / 3 # 3.3333333333333335
10 // 3 # 3
10 % 3 # 1
2 ** 3 # 8
```

◇*Example* :

```
for i in range(len(coin)):
    num = M // coin[i]
    counts.append(num)
    M= M \% coin[i]
```

Data features

```
abs(-5) # 5
round(3.7) # 4, default to 0 decimal places
round(3.14159, 2) # 3.14
min(3, 1, 2) # 1
max(3, 1, 2) # 3
sum([1, 2, 3]) # 6
round(float(number), 2) # transform to float first, round second
```

Loops

▶ Enumerate

```
fruits = ["apple", "banana"]
for i, fruit in enumerate(fruits):
    print(f"{i}: {fruit}") # "0: apple", "1: banana"
```

◇*Example* :

```
# base_count = {
# "A": [1] * n,
# "C": [1] * n,
# "G": [1] * n,
# "T": [1] * n
# }
for index, base in enumerate(motif):
    base_count[base][index] += 1
```

▶ Loop control

```
for i in range(10):
    if i == 3:
        continue # Skip this iteration
    if i == 7:
        break # Exit loop
    print(i)
```

Anonymous function

```
numbers = [1, 2, 3, 4]
squared = list(map(lambda x: x**2, numbers)) # [1, 4, 9, 16]
evens = list(filter(lambda x: x % 2 == 0, numbers)) # [2, 4]
```

Collections

▶ List

```
last = nums.pop() # Pop returns last element
m = [[] for _ in range(len(s))] # construct a 2D list

motif_length= len(motifs[0]) # col numbers of a 2D list
motif_number= len(motifs) # row numbers of a 2D list

list(res) # before printing and sorting a res
```

▶ Tuple

Tuple is often used for coordinates.

```
point = (3, 4)
x, y = point # Basic tuple unpacking
x # 3
y # 4
```

▶ Set

```
a = {1, 2, 3}
b = set([3, 4, 4, 5])
c = set()

a | b # Union: {1, 2, 3, 4, 5}
a & b # Intersection: {3}
```

▶ Dictionary

```
pet = {"name": "Leo", "age": 42}
age = pet.get("age", 0) # Get with default: get(key, default_value)
pet.pop("age") # Remove and return

pet = {"name": "Frieda", "sound": "Bark!"}
pet.keys() # dict_keys(['name', 'sound'])
pet.values() # dict_values(['Frieda', 'Bark!'])
pet.items() # dict_items([('name', 'Frieda'), ('sound', 'Bark!')])

for key in adj_list: # is the same as
for key in adj_list.keys():
if source not in adjacency_list: # is the same as
if source not in adjacency_list.keys():

for m in count:
    if count[m] == ans:
        result.append(m) # append the key only
result.sort()

for x, adj in adj_list.items(): # iterate a dictionary with nested
    ↪ lists
    for y, z in adj:
for neighbor in adjacency_list.get(node, []):
    ↪ dictionary_obj.get(key, default_value)
```

String

▶ String slicing

```
text = "Python"
text[-1] # "n" (last)
text[:3] # "Pyt" (from start)
text[3:] # "hon" (to end)
text[::2] # "Pto" (every 2nd)
text[::-1] # "nohtyP" (reverse)
for i in range(len(seq)): # String rotation
    seq= seq[1:] + seq[:1]
s = s_2[1:] # remove the first character of a string
s = s_1[:-1] # remove the last character of a string
```

▶ String patching

```
greeting = "me" + "ow!" # "meow!"
seq= input() + "$" # BWT encoding
repeat = "Meow!" * 3 # "Meow!Meow!Meow!"
length = len("Python") # 6
```

▶ String methods

```
"a".upper() # "A"
"A".lower() # "a"
"  a  ".strip() # "a"
"abc".replace("bc", "ha") # "aha"
"a b".split() # ["a", "b"]
"-".join(["a", "b"]) # "a-b"
```

▶ String formatting

```
name = "Aubrey"
age = 2
f"{name} is {age} years old" # "Aubrey is 2 years old"
```

▶ String to List

String type cannot be altered, so convert to a list first and alter the list, then ultimately patch them.

```
pattern = "AGCGGAG"
neighbor = list(pattern) # ['A', 'G', 'C', 'G', 'G', 'A', 'G']
pattern = ''.join(neighbor) # "AGCGGAG"
```

General constructs

```
a, b = b, a # Swap values

matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
flat = [item for sublist in matrix for item in sublist] # Flatten a
    ↪ list of lists

unique_unordered = list(set(my_list)) # Remove duplicates, unordered

from collections import Counter
counts = Counter(my_list) # Count occurrences

import heapq # remember to import first
pq= []
heapq.heappush(pq, (0.0, s))
heapq.heappush(pq, (newDist, i))
_, pos= heapq.heappop(pq)

from itertools import product
for indices in product(range(k), repeat=d):
    for replacements in product('ACGT', repeat=d):
        neighbor= list(pattern)
        for idx, replacement in zip(indices, replacements):
            neighbor[idx]= replacement
```

Numpy

IO

a = np.loadtxt("myfile.txt") a = np.loadtxt(sys.stdin)

Creating array

▶ With specified numbers

```
a = np.array([1,2,3])
b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
```

▶ With placeholders

```
a = np.empty((3,2)) # Array full of uninitialized values
a = np.zeros((3,4)) # Array full of zeros
a = np.ones((2,3,4), dtype = int) # Array full of ones
a = np.full((2,3), 7) # Array full of constants
a = np.random.random((2,2)) # Array full of random values
```

▶ Filling an array

▷*With some specified values:*

```
a = np.full((nodemax, nodemax), float('-inf')) # Array full of
    ↪ negative infinity
np.fill_diagonal(a, float('-inf')) # Fill the diagonal with infinity
```

▷*Two equivalent ways to create and fill a 2D array:*

```
rocks = np.ndarray((n + 1, m + 1), dtype=bool)
rocks.fill(False) # equals to
rocks = np.full((n + 1, m + 1), False, dtype=bool)
```

▷*Fill the array based on what has been done:*

```
for i in range(1, n + 1):
    rocks[i, 0]= not rocks[i- 1, 0]

if all((rocks[i- 1, j- 1], rocks[i, j- 1], rocks[i- 1, j])):
    rocks[i, j]= False
```

Inspecting array

```
b.shape # dimensions
row_number, col_number = b.shape
row_number = b.shape[0]
col_number = b.shape[1]
len(a) # length of the first dimension
a.size # total number of elements
```

Arithmetic operations

```
g = a - b # subtraction
h = a + b # addition
b = sqrt(a) # square root
```

Comparison

```
a == b # returns a boolean array of comparison results
a < 2 # returns a boolean array of comparison results
```

Data features

```
a.sum() # 1D array-wise sum
a.mean() # mean of all elements in 1D array
a.median() # mean of all elements in 1D array
b.min(axis = 0) # column-wise minimum
b.max(axis = 1) # row-wise maximum
```

Sorting

```
a.sort() # sort 1D array
b.sort(axis = 1) # sort each row
```

Slicing

▶ 1D array

```
a[2] # just like list
a[::-1] # reversion
```

▶ 2D array

```
b[1,2] # a single data point
b[0:2, 1] # first two rows of the second column
```

```
for index, row in raw.iterrows(): # access elements in numpy 3D
    ↪ array is just like how to access elements in numpy dataframe
    array[row['source'], row['target']]= row['weight']
    array[row['target'], row['source']]= row['weight']
```

Pandas

IO

```
df = pd.read_csv(sys.stdin)
```

Iteration

```
for index, row in raw.iterrows():
    array[row['source'], row['target']]= row['weight']
    array[row['target'], row['source']]= row['weight']
```

Slicing

```
df[0:] # all rows except the first
list = dataframe['source'] # get column as a list
```

Data features

```
df.sum() #Sum of values
df.cumsum() # Cummulative sum of values
df.min()/df.max() # Minimum/maximum values
df.mean() # Mean of all values
df.median() # Median of all values
nodemax= max(raw['source'].max(), raw['target'].max()) + 1 # two
    ↪ ways of using max
```

Operations

```
df.drop(columns=['cell_type', 'sample'], axis = 1) # Dropping
df.sort_values(by="Country") # Sorting
```

Information

```
df.shape # (rows, columns)
df.count() # Number of non-NA values
```

Applying functions

```
s = s.max() - s.min()
df.apply(s) # Apply function column wise

f = lambda x: x*2
df.applymap(f) # Apply function element wise
```