Python for the Life Sciences
SYNTAX CHEATSHEET

Variable Assignment

int	my_int = 7
float	my_float = 7.0
string	my_string = 'Kathmandu'
list	my_list = ['Kathmandu', 'Jinja', 'Accra', 'Whistler']
dict	my_dict = { 'key1': 123, 'key2': 37, 321: 'value' }
set	my_set={'apple', 'orange', 'banana', 'pear'}
tuple	my_tuple = ('bcl4', 786, 5, 'Brian', 2.09)

Arithmetic Operators

Addition	a+b
Subtraction	a-b
Multiplication	a*b
Division	a/b
Modulo	a%b
Exponent	a**b

Type Conversion

int()	to integer
float()	to float
str()	to string
list()	to list
set()	to set

Comparison, membership and identity

==	is equal to
!=	is not equal to
>	is greater than
<	is less than
>=	is greater than or equal to
<=	is less than or equal to
is None	is null
is not None	is not null
in	string, list or dict(keys) contains e.g. the following return true so can be tested by conditional statement: "GCT" in "GTACGCTTAG" #true 5 in [1,2,5,7,9] #true "be" in ["ant","bee","cat"] #false "bee" in ["ant","bee","cat"] #true
not in	string, list or dict(keys) does not contain..

String formatting

f-strings	area=3021 #square km
	trees_per_sqkm=80000
	print(f'There are {area*trees_per_sqkm} trees in {area_km2} km2')	

Decimal places	from math import pi f'{pi:.2f}' f'{pi:+.2f}'	Output: 3.14 +3.14 (+/-sign)
Format percentage	var=0.1 f'{var:.2%}'	10.00%
Scientific notation	var=1000 f'{var:.2e}'	1.00e+03

Functions

```
def add_numbers(a,b):
    '''This function adds two numbers'''
    return a+b
```

Escape characters

\n	New line
\t	Tab

Note

Many of the functions and methods describe here take optional additional arguments, such as start and finish points. These are described in the official Python documentation.

String Manipulation

+	Concatenation	my_string = "Hello " + "World!"
str[index]	String indices	str[0] #first character str[2:5] #3rd to 5th character str[::-1] #reverse string
str.rstrip(chars)	Strip char(s) from right	str.rstrip("\n")
str.split(delim)	Split on delimiter, creating list	"Berlin,Delhi,Moscow".split(",") #['Berlin', 'Delhi', 'Moscow']
.lower()	Convert to lower case	"HELLO".lower() #"hello"
.upper()	Convert to UPPER CASE	"hello".upper() #"HELLO"
len()	Length of string	len("ATGCTA") #6
.replace(old,new)	Replace old with new	"ATGTCG".replace("T","U") #AUGUCG
.find(substring)	Find first occurrence of substring	"ATGTCG".find("TC") #3

.join(seq)	Join any iterable (string, list, etc) using delimiter. Result is a string.	seq = ("abc")
		print("-".join(seq))
		# a-b-c

Lists

list[index]	List indices	my_list[0] #first item my_list[2:5] #3rd to 5th items my_list[-1] #last item
list.sort()	Sorts list in (alpha)numerical order, in-place,	
sorted(list)	Function returns a sorted list (original unmodified)	
+	Concatenates two lists	[1,2] + [3,4] # gives [1,2,3,4]
enumerate()	Loops over a list with indices	for i, item in enumerate(list): print(i , item) #index, item
.join()	Join list items using a delimiter (also works on other sequences)	seq = ("a", "g", "t") print(",".join(seq)) #prints a,g,t
len()	Number of items in list	len(seq) #returns 3
.append()	Add items to end of list	seq.append("c")
.extend()	Add list items to end of list	num = [1,2] num.append([3,4]) #num is now [1,2,3,4]
.pop()	Removes and returns the last item from a list	myList = ["dog", "donkey", 'lemur', 'gorilla']; print(myList.pop()) #Prints "gorilla" and removes it from myList print(myList.pop()) #Prints "lemur" and removes if from myList
del	Remove a item at given index	myList = ['dog', 'donkey', 'lemur', 'gorilla']; del myList[2] # delete index 2 del mylist[2:3] # or use slice notation

.remove()	Remove the first item with given value	myList = ['dog', 'donkey', 'lemur', 'gorilla'];
		myList.remove('donkey')

.join()	Join any iterable (string, list, etc) using delimiter. Result is a string.	my_list = [a,b,c] print("-".join(my_list)) # a-b-c

Dictionaries

sorted()	Function returns a sorted list of dictionary keys	
dict[key]	Returns the value associated with 'key' in the dictionary.	my_dict['ATGGTA'] # value for 'ATGGTA' key
my_dict.keys()	Returns all keys in the dictionary as a list	
my_dict.values()	Returns all values in the dictionary as a list	

my_dict.items()	Iterator that returns items in a dictionary as list of (key, value) tuples.	for (k,v) in my_dict.items():
		    print(f"Key is {k}, Value is {v}")

my_dict.get(key, default_value)	Return value associated wth key. Allows you to provide a default value if the key is missing, avoiding key errors (default is None)	

Sets

.add()	Adds a single item to set	myset.add(1)
.update()	Adds one or more iterables to a set	myset.update([1, 2], [i for i in range(3, 5)]) #{1, 2, 3, 4}

Convert List to Set	cities = set(['Paris', 'Berlin', 'London','Berlin','Paris'])
	print(cities) # prints {'London', 'Berlin', 'Paris'}	

len()	Number of items in set	len(my_set) #returns 4

Loops

```
for i in range (1,29,3):
    print(i) #counts from 1 to 28, step size 3
```

```
for dog in ["Collie", "Poodle", "Spaniel", "Labrador"]:
    print (dog)
```

```
while i<=10:
    print (i)
    i += 1
```

```
while True:
    n = raw_input("Please enter 'hello':")
    if n.strip() == 'hello':
        break
```

Conditional Statements

```
if my_integer==7:
    print("Integer is 7")
elif my_integer>7:
    print ("Integer greater than 7")
else:
    print("Integer is less than 7")
```

```
if "GATATC" in "ATGTAAGATATCTAG":
    print("EcoRV site found in DNA sequence")
else:
    print("EcoRV site not present in DNA sequence")
```

List Comprehensions

simple	[x * x for x in range(5)]
if	[3*x for x in vec if x > 3]

zip	l1=1,2,3,4,5
	l2=6,7,8,9,0
	[k*v for k,v in zip(l1,l2)]
	#[6, 14, 24, 36, 0]

nested	mat = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] inv=[[row[i] for row in mat] for i in range(len(mat[0]))] # Above list comprehension inverts the matrix # [[1, 4, 7], [2, 5, 8], [3, 6, 9]]

File Operations

Opening files using with statement

```
with open('Mus_musculus.fa') as f:
    for line in f:
        if line.startswith(">"):
            #other processing of line
```

Opening and Writing Gzipped Files

```
import gzip
with gzip.open('Mus_musculus.fa.gz','rt') as f: #rt is text mode, r would be binary
    for line in f:
        #other processing of line

with gzip.open('output.txt.gz', 'wt') as outfile: #wt writes in text mode – wb for binary
    outfile.write("Test to Write" )
```

Selecting all files in a directory:

```
import os
files = [f for f in os.listdir('./') if re.match(r'^.*\.txt$', f)]
for file in files:
    #other processing of file
```

```
import os
for file in os.listdir("./mydir"):
    if file.endswith(".jpg"):
        print(file," is a JPEG")
```

File opening options

r	Open a file for reading.
w	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
x	Open a file for exclusive creation. If the file already exists, the operation fails.
a	Open for appending at the end of the file without truncating it. Creates a new file if it does not exist.
b	Open in binary mode
t	Open in text mode
+	Open file for updating (reading or writing)

Zipping and unpacking

list(zip(list1,list2))	Zips one or more lists together	list1 = [1,2,3]
		list2 = ['a','b','c']
		combined=list(zip(list1, list2))
		print(combined) #[(1, 'a'), (2, 'b'), (3, 'c')]

dict(zip(keyList, valList))	Zips two lists into a dict	l1=1,2,3,4,5
		l2=6,7,8,9,0
		print(dict(zip(l1,l2)))
		# {1: 6, 2: 7, 3: 8, 4: 9, 5: 0}

Unzipping with zip...	* unpacks arguments	#following on from above
		l1, l2 = zip(*combined)
		print(l1) #(1, 2, 3)
		print(l2) #('a', 'b', 'c')

Regular Expressions Methods

re.match()	Search only from beginning of string. Returns match object	re.match(r'\d','7A1B2C3D4E5F6') #<_sre.SRE_Match object; span=(0, 1), match='7'>
re.fullmatch()	Match if whole string matches pattern. Returns match object.	re.fullmatch(r'\d','A1B2C3D4E5F6') #None (not a full match)
re.search()	Find 1st occurrence, anywhere in the string. Returns match object.	re.search(r'\d','A1B2C3D4E5F6') #<_sre.SRE_Match object; span=(1, 2), match='1'>
re.findall()	Returns all the non-overlapping matches of patterns in a string as a list of strings.	re.findall(r'\d','A1B2C3D4E5F6') # ['1', '2', '3', '4', '5', '6']
re.finditer()	returns iterator yielding MatchObject instances over all non-overlapping matches	numbers = [match.group() for match in re.finditer(r'\d','A1B2C3D4E5F6')] ['1', '2', '3', '4', '5', '6']
re.split()	Splits the string by occurrence of a pattern.	re.split("[a-z]","91w27e32s89") # ['91', '27', '32', '89']

System Commands (import os, shutil)

os.system(system_command)	Run at command line
os.mkdir(dir)	Create a directory,
os.rename(new_dir, old_dir	Rename a directory
os.listdir("directory")	List directory contents
os.chdir(dir)	Change directory
os.rmdir(dir)	Remove directory
shutil.rmtree(dir)	Remove directory (including files)

Match objects

match.group(0)– holds whole match object
match.group(1)-first match group

```
import re
line = "Cats are smarter than dogs"
matchObj = re.match( r'(.*) are (.*?) .*', line, re.M|re.I)
if matchObj:
   print "matchObj.group() : ", matchObj.group()
   print "matchObj.group(1) : ", matchObj.group(1)
   print "matchObj.group(2) : ", matchObj.group(2)
else:
   print "No match!!"
```

Regex Patterns

Too many for this sheet – see dedicated regex cheatsheet

args and kwargs

```
def my_function(*args, **kwargs):
    print(f"Unnamed args: {args}") #args is tuple
    print(f"Keyword args: {kwargs}") #kwargs is list

my_function("Australopithicus", 13, age=14, height=1.83)
```

```
#Output:
Unnamed args:
('Australopithicus', 13)
Keyword args: {'height': 1.83,
'age': 14}
```