CS 5 Green

Learning Goals

- Explain benefits of modular programming
- Practice modular programming

Who is the ‘audience’ for your code?

- Computer
-

Fizkes, shutterstock.com

docstring format matters

```
def docstringIsComment():
    # This function prints Hello World
    print("Hello world")

''' This function prints Hello World '''
def docstringTooEarly():
    print("Hello world")

def docstringTooLate():
    print("Hello world")
    ''' This function prints Hello World '''

def docstringJustRight():
    ''' This function prints Hello World '''
    print("Hello world")
```

```
help(function_name)
```

Review: the virtues of negative thinking!

0	1	2	3	4
-5	-4	-3	-2	-1

```
my_list = [42, 47, 23, [3.141, 2.718], 5]

>>> my_list[len(my_list)-3]
23

>>> my_list[-3]
23

>>> my_list[-2:]
[[3.141, 2.718], 5]
```

Review: the virtues of negative thinking!

```
                0123456789
my_string = "I luv spam"

>>> my_string[:-1]
'I luv spa'

>>> my_string[-3:]
'pam'
```

Review: two types of for loop

```
def bee_ify(lst):
    '''Add ”bee " to every string in list L.'''
    new_lst = []
    for s in lst:
        new_lst.append(”bee ” + s)
    return new_lst
```

For loop goes directly over the list L

```
>>> bee_ify([”kind", ”brave", ”happy”])
[‘bee kind’, ‘bee brave', ‘bee happy']
```

Review: two types of for loop

```
def spam_count(S):
    '''Count occurrences of "spam" in input S.'''
    counter = 0
    for i in range(len(S)):
        if S[i:i+4] == "spam":
            counter = counter + 1
    return counter
```

For loop goes over indices into string S

[0, 1, 2, … len(S)-1]

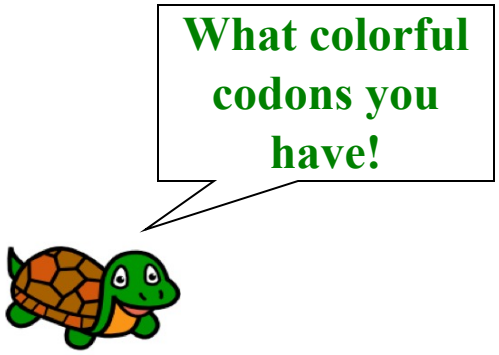
```
>>> spam_count("gspamtspammspamn")
3
```

An alternate use of `in`

```
>>> for num in range(1, 100): # what we've seen before
...

>>> 42 in [3, 67, 42, 18, 2001] # new use
True

>>> 42 in [13, 33, 300]
False

>>> food = ["carrots", "coffee", "arugula", "spam"]
>>> if "spam" in food: print("Yay!!!")
...
Yay!!!

>>> "bio" in "symbiont"
True
```

Displaying output for the user: the `print` function

```
def print_codons(DNAstring):
    for i in range(0, len(DNAstring), 3):
        print("Next codon: ", DNAstring[i:i+3])
    # no return statement necessary!

                    0123456789
>>> print_codons("AAATTTGGGC")
Next codon: AAA
Next codon: TTT
Next codon: GGG
Next codon: C
```



return vs print...


```
def dbl(x):
    return 2 * x


def happy(input):
    y = dbl(input)
    return 2 * y


>>> happy(4)
16
```

```
def trbl(x):
    print(2 * x)
    return


def sad(input):
    y = trbl(input)
    return 2 * y


>>> sad(4)
8
TypeError: unsupported
operand type(s) for *:
'int' and 'NoneType'
```

implicit return

Advantages of modular programming

- Simpler to read and understand
- Easier to test and debug
- Easier to modify in future
- Easier to reuse parts of the code

www.modular.org

Worksheet

standard_dev.py

(Modularity and top-down design!)

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

- Step 1: Find the expected value (mean).
- Step 2: For all data values, find the square of their distance to the mean (squared residuals).
- Step 3: Sum the values from Step 2 (sum of squared residuals).
- Step 4: Divide by the number of data values minus 1 (variance).
- Step 5: Take the square root (standard deviation).
- Let's write the function signatures and docstrings!

Professor P.I. Pette: calculate median genome size in intracellular pathogens vs. free livers

speciesList	habitatList	genomeSizeList
"Agrobacterium tumefaciens C58"	["free living"]	5100
"Chlamidia psittaci 1H"	["intracellular"]	1450
"Salmonella typhimurium"	["intracellular", "free living"]	4780
"Lactococcus lactis C2"	["free living"]	2500
"Escherichia coli K12"	["free living","commensal"]	4670

http://www.genomesize.com/prokaryotes/table1/

The data (in Python)

```
habitat_list = [["free living"],
                ["intracellular"],
                ["intracellular", "free living"],
                ["free living"],
                ["free living", "commensal"],
                ["intracellular"],
                ["free living", "commensal"],
                ["free living"],
                ["free living"],
                ["free living"],
                ["free living", "commensal"],
                ["free living"],
                ["intracellular"],
                ["free living", "commensal"],
                ["free living", "commensal"]]
```

```
genome_size_list = [5100,
                    1450,
                    4780,
                    2500,
                    4670,
                    1600,
                    4140,
                    4330,
                    6400,
                    6700,
                    4592,
                    3100,
                    1120,
                    7490,
                    4200]
```

Nick's code (calculates median genome size in intracellular pathogens vs. free livers)

```
from bacGenData import *

def gsize(habitat_list, genome_size_list):
    a = []
    for i in range(len(genome_size_list)):
        if 'intracellular' in habitat_list[i]:
            a.append(genome_size_list[i])
    a.sort()
    if len(a) % 2 != 0:
        mid_index = int((len(a) - 1) / 2)
        print('intracellular', 1.0 * a[mid_index])
    else:
        y = int((len(a) / 2) -1)
        b = int(len(a) / 2)
        print('intracellular', (a[y] + a[b]) / 2.0)
    z = []
    for i in range(len(genome_size_list)):
        if 'free living' in habitat_list[i]:
            z.append(genome_size_list[i])
    z.sort()
    if len(z) % 2 != 0:
        mid_index = int((len(z) - 1) / 2)
        print('free living', 1.0 * z[mid_index])
    else:
        y = int((len(z) / 2) -1)
        b = int(len(z) / 2)
        print('free living', (z[y] + z[b]) / 2.0)
```

Prof Pette now wants to look at **exclusively** intracellular or free living species.
But Nick is out of town...

```
def gsize(habitat_list, genome_size_list):
    '''Calculate and print median genome size in intracellular pathogens
    and also in free livers.'''

    intra_sizes = find_sizes("intracellular", habitat_list, genome_size_list)
    free_sizes = find_sizes("free living", habitat_list, genome_size_list)

    print("intracellular", median(intra_sizes))
    print("free living", median(free_sizes))
```

median

```
def median(L):
    '''Calculates the median of list L.'''
    L.sort()
    if len(L) % 2 != 0:
        # odd number of elements, take middle one
        mid_index = int((len(L) - 1) / 2)
        return 1.0 * L[mid_index]
    else:
        # even number of elements
        mid_index1 = int((len(L) / 2) -1)
        mid_index2 = int(len(L) / 2)
        return (L[mid_index1] + L[mid_index2]) / 2.0
```

[2,7,18,23,500]

[2,7,18,23]

```
>>> median([500,18,2,23,7])
18.0
```

Notes

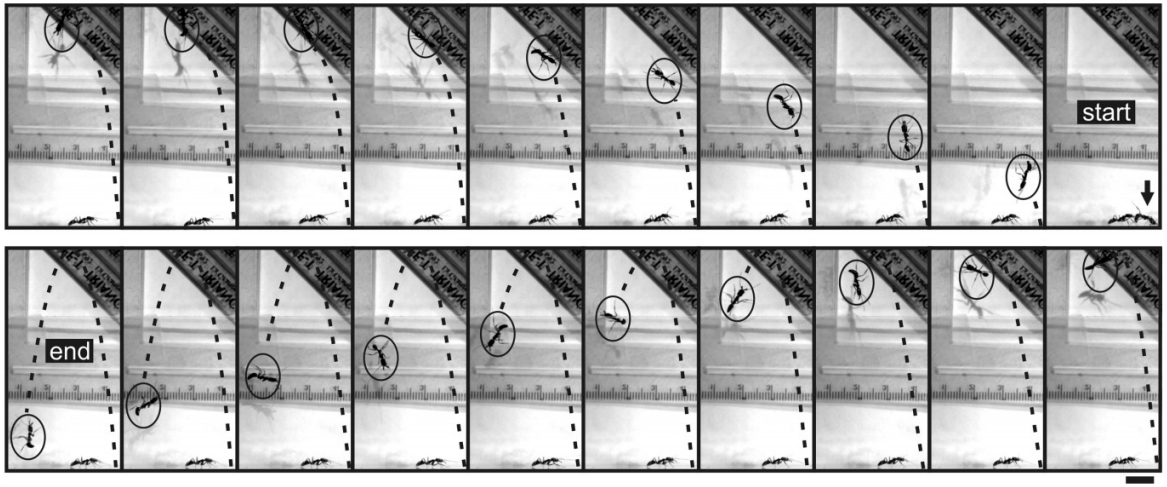

```
def find_sizes(habitat, habitat_list, genome_size_list):
    '''Find all species which live in habitat, and return a list of
    their genome sizes.'''

    out_list = []









    return out_list
```

```
>>> find_sizes("commensal", habitat_list, genome_size_list)
[4670, 4140, 4592, 7490, 4200]
```

Finding extremz!

```
dictionary = [
    "abdomen",
    "abdominal",
    "abduct",
    "abduction,"
    "aberration,"
    "abet,"
    "abhor,"
    "abhorrence,"
    "abhorrent,"
    "abide,"
    "abiding,"
    "ability,"
    "abject,"
    "ablaze,"
  ...
  ...
  etc.
  ...
  ...]
```


```
def z(input):
    '''Count z's in a string'''
    counter = 0
    for symbol in input:
        if symbol == 'z':
            counter = counter +1
    return counter
```



Alex Wild; Patek et al. (2014) *PNAS*

Notes


```
def extremz(word_list):
    '''Find and return the word with the most z's'''
```

Name _____

Worksheet

standard_dev.py

(Modularity and top-down design!)

Let's write the function signatures and docstrings!