(1) Multi-Way If Statement

In [ ]:

```
'''
Write function BMI() that:

(1) Takes as input a person’s height (in inches) and weight (in pounds); and
(2) Computes the person’s BMI and prints an assessment, as shown below.

>>> BMI(190, 75)
Normal
>>> BMI(140, 75)
Underweight
>>> BMI(240, 75)
Overweight

The function does not return anything.

The Body Mass Index is the value (weight * 703) / (height ** 2).
Indexes below 18.5 or above 25.0 are assessed as underweight and overweight, respective
indexes in between are considered normal.
'''
```

In [9]:

```
def BMI(w, h):
    i = (w * 703) / (h**2)
    if i >= 25.0:
        print('Overweight')
    elif i > 18.5:
        print ('Normal')
    else:
        print('Underweight')
```

In [10]:

```
BMI(190, 75)
```

```
Normal
```

In [11]:

```
BMI(140, 75)
```

```
Underweight
```

In [12]:

```
BMI(240, 75)
```

```
Overweight
```

In [13]:

```
def BMI2(w, h):
    i = (w * 703) / (h**2)
    if i <= 18.5:
        print('Underweight')
    elif i < 25.0:
        print ('Normal')
    else:
        print('Overweight')
```

In [14]:

```
BMI2(190, 75)
```

```
Normal
```

In [15]:

```
BMI2(140, 75)
```

```
Underweight
```

In [16]:

```
BMI2(240, 75)
```

```
Overweight
```

In [17]:

```
# Example of a Failing Multi-Way If Statement
def BMI3(w, h):
    i = (w * 703) / (h**2)
    if i > 18.5:
        print('Normal')
    elif i >= 25.0:
        print ('Overweight')
    else:
        print('Underweight')
```

In [18]:

```
BMI3(190, 75)
```

```
Normal
```

In [19]:

```
BMI3(140, 75)
```

```
Underweight
```

In [20]:

```
BMI3(240, 75) # Fails here. Should return "Overweight".
```

```
Normal
```

(2) Return vs. Print Inside of the If Statement In the For Loop

In [21]:

```
def test(n):
    for i in range(n):
        if i > 5:
            print(i) # print() will print until the end of the range; not once the stat
        else:
            print('wrong')
```

In [22]:

```
test(10)
```

```
wrong
wrong
wrong
wrong
```

```
wrong
wrong
6
7
8
9
```

In [23]:

```
def test(n):
    for i in range(n):
        if i > 5:
            return i # return will stop the iteration when the statement becomes true.
        else:
            print('wrong')
```

In [24]:

```
test(10)
```

```
wrong
wrong
wrong
wrong
wrong
wrong
```

Out[24]:

```
6
```

In [25]:

```
def test(n):
    for i in range(n):
        if i > 5:
            return i
        else:
            return 'wrong'
```

In [26]:

```
test(10) # Evaluates for when n = 0 value is false. This breaks loop immediately, retur
```

Out[26]:

```
'wrong'
```

In [27]:

```
def test(n):
    for i in range(n):
        if i > 5:
            print(i)
        else:
            return i
```

In [28]:

```
test(10) # If is false, so evaluates for else, returning the value 0
```

Out[28]:

```
0
```

In [29]:

```
def test(n):
    for i in range(n):
        if i > 5:
            print(i)
        return i
```

In [30]:

```
test(10)
```

Out[30]:

```
0
```

In [31]:

```
def test(n):
    for i in range(n):
        if i > 5:
            print(i)
    return i
# Evalautes Inner Loop First. Is True When n = 6, 7, 8, 9
# After the Inner Loop Is Complete, the Outer Loop Runs, Which Returns the Last Value o
```

In [32]:

```
test(10)
```

```
6
7
8
9
```

Out[32]:

```
9
```

(3) Loop Pattern: Iteration vs. Counter

In [ ]:

```
'''
Develop function checkSorted() that:

(1) Takes a list of comparable items as input; and
(2) Returns True if the sequence is increasing and False, otherwise.

>>> checkSorted([2, 4, 6, 8, 10])
True
>>> checkSorted([2, 4, 6, 3, 10])
False
>>>
'''
```

In [33]:

```
def checkSorted(lst):
    for i in range(len(lst) - 1):
        if lst[i] >= lst[i + 1]:
            return False
    return True
```

In [34]:

```
checkSorted([2, 4, 6, 8, 10])
```

Out[34]:

```
True
```

In [35]:

```
checkSorted([2, 4, 6, 3, 10])
```

Out[35]:

```
False
```

In [ ]:

```
'''
Write function arithmetic() that:

(1) Takes as input a list of numbers; and
(2) Returns True if the numbers in the list form an arithmetic sequence and False, othe

>>> arithmetic([3, 6, 9, 12, 15])
True
>>> arithmetic([3, 6, 9, 11, 14])
False
>>> arithmetic([3])
True
'''
```

In [36]:

```
def arithmetic(lst):
    for i in range(len(lst) - 2):
        if lst[i + 1] - lst[i] != lst[i + 2] - lst[i + 1]:
            return False
    return True
```

In [37]:

```
def arithmetic(lst):
    if len(lst) < 3:
        return True
    for i in range(len(lst) - 2):
        if lst[i + 1] - len(lst[i]):
            return False
        return True
```

In [38]:

```
def arithmetic(lst):
    if len(lst) < 3:
        return True
    diff = lst[1] - lst[0]
    for i in range(len(lst) - 1):
        if lst[i + 1] - lst[i] != diff:
            return False
    return True
```

In [39]:

```
arithmetic([3, 6, 9, 12, 15])
```

Out[39]:

```
True
```

In [40]:

```
arithmetic([3, 6, 9, 11, 14])
```

Out[40]:

```
False
```

In [41]:

```
arithmetic([3])
```

Out[41]:

```
True
```

(4) Accumulator Loop Pattern

In [ ]:

```
'''
Write function factorial() that:

(1) Takes a non-negative integer n as an input; and
(2) Returns n.

n! = n x (n-1) x (n-2) x ... x 3 x 2 1
0! = 1

>>> factorial(0)
1
>>> factorial(1)
1
>>> factorial(3)
6
>>> factorial(6)
720
'''
```

In [42]:

```
def factorial(n):
    res = 1
    for i in range(1, n + 1):
        res *= i
    return res
```

In [43]:

```
factorial(0)
```

Out[43]:

```
1
```

In [44]:

```
factorial(1)
```

Out[44]:

```
1
```

In [45]:

```
factorial(3)
```

Out[45]:

```
6
```

In [46]:

```
factorial(6)
```

Out[46]:

```
720
```

In [ ]:

```
'''
Write function acronym() that:

(1) Takes a phrase (i.e., a string) as and input; and
(2) Returns the acronym for the phrase.
```

```
>>> acronym('Random access memory')
'RAM'
>>> acronym("GNU's not UNIX")
'GNU'
'''
```

In [47]:

```
def acronym(s):
    lst = s.split()
    res = ''
    for i in lst:
        res += i[0].upper()
    return res
```

In [48]:

```
acronym('Random access memory')
```

Out[48]:

```
'RAM'
```

In [49]:

```
acronym("GNU's not UNIX")
```

Out[49]:

```
'GNU'
```

In [ ]:

```
'''
Write function divisors() that:

(1) Takes a positive integer n as input; and
(2) Returns the list of positive divisors of n.

>>> divisors(1)
[1]
>>> divisors(6)
[1, 2, 3, 6]
>>> divisors(11)
[1, 11]
'''
```

In [50]:

```
def divisors(n):
    lst = []
    for i in range(1, n + 1):
        if n % i == 0:
            lst.append(i)
    return lst
```

In [51]:

```
divisors(1)
```

Out[51]:

```
[1]
```

In [52]:

```
divisors(6)
```

Out[52]:

```
[1, 2, 3, 6]
```

In [53]:

```
divisors(11)
```

Out[53]:

```
[1, 11]
```

(5) Nested For Loop

In [ ]:

```
'''
Write function inBoth() that takes:

(1) 2 lists as input; and
(2) Returns True if there is an item that is common to both lists and False, otherwise.

>>> inBoth([3, 2, 5, 4, 7], [9, 0, 1, 3])
True
>>> inBoth([2, 5, 4, 7], [9, 0, 1, 3])
False
'''
```

In [54]:

```
def inBoth(lst_1, lst_2):
    for i in lst_1:
        for j in lst_2:
            if i == j:
                return True
    return False
```

In [55]:

```
inBoth([3, 2, 5, 4, 7], [9, 0, 1, 3])
```

Out[55]:

```
True
```

In [56]:

```
inBoth([2, 5, 4, 7], [9, 0, 1, 3])
```

Out[56]:

```
False
```

In [ ]:

```
'''
Write function pairSum() that takes as input:

(1) A list of numbers;
(2) A target value; and
(3) Prints the indexes of all pairs of values in the list that add up to the target val

>>> pairSum([7, 8, 5, 3, 4, 6], 11)
0 4
1 3
2 5
3 1
4 0
5 2
'''
```

In [57]:

```
def pairSum(lst, t):
    for i in lst:
        for j in lst:
            if i + j == t:
                print(lst.index(i), lst.index(j))
```

In [58]:

```
pairSum([7, 8, 5, 3, 4, 6], 11)
```

```
0 4
1 3
2 5
3 1
4 0
5 2
```

In [ ]:

```
'''
Implement function pixels() that takes as input:

(1) A two-dimensional list of nonnegative integer entries (representing the values of p
(2) Returns the number of entries that are positive (i.e., the number of pixels that ar
The function should work on two-dimensional lists of any size.

>>> lst = [[0, 156, 0, 0], [34, 0, 0, 0], [23, 123, 0, 34]]
>>> pixels(lst)
5
>>> lst = [[123, 56, 255], [34, 0, 0], [23, 123, 0], [3, 0, 0]]
>>> pixels(lst)
7
'''
```

In [59]:

```
def pixels(lst):
    res = 0
    for col in lst:
        for i in col:
            if i > 0:
                res += 1
    return res
```

In [60]:

```
lst = [[0, 156, 0, 0], [34, 0, 0, 0], [23, 123, 0, 34]]
pixels(lst)
```

Out[60]:

```
5
```

In [61]:

```
lst = [[123, 56, 255], [34, 0, 0], [23, 123, 0], [3, 0, 0]]
pixels(lst)
```

Out[61]:

```
7
```

(6) While Loop

In [ ]:

```
'''
Write a function fibonnaci() that:

(1) Takes as input a bound; and
(2) Returns the first Fibnonaci number greater than the bound.

Fibonnaci sequence
1, 1, 2, 3, 5, 8, 13, ...
'''
```

In [62]:

```
def fibonnaci(n):
    a = 1
    b = 1
    c = a + b
    while c <= n:
        a = b
        b = c
        c = a + b
    return c
```

In [63]:

```
fibonnaci(10)
```

Out[63]:

```
13
```

(7) Infinite Loop Pattern

In [ ]:

```
'''
Write a function hello2() that:

(1) Repeatedly requests the name of the user; and
(2) Then greets the user.
'''
```

In [64]:

```
def hello2():
    while True:
        name = input('Your name please:')
        print('Hello, {}'.format(name))
```

In [ ]:

```
hello2()
```

(8) Break and Continue Statements

In [68]:

```
table = [[2, 3, 0, 6],
         [0, 3, 4, 5],
         [4, 5, 6, 0]]
```

In [ ]:

```
'''
Write function before0() that:

(1) Takes a 2-D list of numbers as and input; and
(2) Prints a 2-D table of numbers of the 2-D list. If there is 0 in the list, numbers a
'''
```

In [69]:

```
def before0(lst):
    for c in lst:
        for n in c:
            if n == 0:
                break
            print(n, end = ' ')
        print()
```

In [70]:

```
before0(table)
```

```
2 3

4 5 6
```

In [ ]:

```
'''
Write function ignore0() that:

(1) Takes a 2-D list of numbers as an input; and
(2) Prints 2-D table of numbers in the 2-D list. If there is 0 in the list, it will not
'''
```

In [71]:

```
def ignore0(lst):
    for c in lst:
        for n in c:
            if n == 0:
                continue
            print(n, end = ' ')
        print()
```

In [72]:

```
def ignore0(lst):
    for c in lst:
        for n in c:
            if n != 0:
                print(n, end = ' ')
        print()
```

In [73]:

```
ignore0(table)
```

```
2 3 6
3 4 5
4 5 6
```

In [ ]:

```
'''
Write function is_prime() that:
```

```
(1) Takes a positive integer n as input; and
(2) Returns True if n is a prime number and returns False, otherwise.

>>> is_prime(2)
True
>>> is_prime(6)
False
>>> is_prime(11)
True
'''
```

In [74]:

```
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

In [75]:

```
is_prime(2)
```

Out[75]:

```
True
```

In [76]:

```
is_prime(6)
```

Out[76]:

```
False
```

In [77]:

```
is_prime(11)
```

Out[77]:

```
True
```

In [ ]:

```
'''
Write function find_largest_prime() that:

(1) Takes a positive integer n as an input; and
(2) Returns the largest prime number that is smaller than n.
'''
```

In [78]:

```
def find_largest_prime(n):
    res = 2
    for i in range (2, n):
        if is_prime(i) == True:
            res = i
    return res
```

In [79]:

```
find_largest_prime(100)
```

Out[79]:

```
97
```

In [80]:

```
find_largest_prime(100000)
```

Out[80]:

```
99991
```

In [81]:

```
def find_largest_prime_2(n):

    def is_prime(n):
        if n < 2:
            return False
        for i in range(2, n):
            if n % i == 0:
                return False
        return True

    for i in range(n - 1, 1, -1):
        if is_prime(i) == True:
            return i
```

In [82]:

```
find_largest_prime_2(10000000)
```

Out[82]:

```
9999991
```