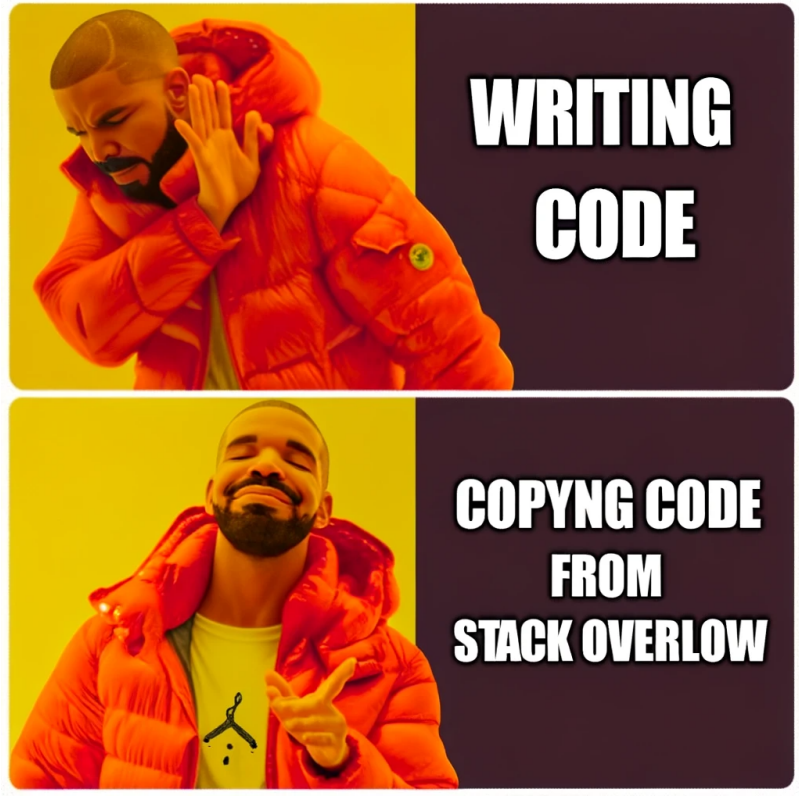
Discussion 2

DSC 20, Fall 2023

Meme of the week



Courtesy of DALL-E

Did you enjoy this week's meme?

A. Yes

B. No

C. What's a meme

D. It's pronouncd /mi-mi/ not /mēm/

Agenda

- imports
- escape character / following style
- **Content**
    - loops
    - Mutability
    - lists, tuples, sets
    - in-place vs not
    - Indexing
    - List Methods
    - String Methods
- Practice Questions

About Imports

"import" is another special keyword in python that has a unique function - it makes code from one body available in another. This can be something like a function from a different file or a whole package (ex. Pandas)

- We "ban" import in this class because many packages are too powerful (and out of the scope of this course)
    - ex. If we asked you to calculate Mean Squared Error on a dataset, you could just import a package for it
- When an import is necessary, we will explicitly import the package/module for you in the starter

Packages are powerful tools that you'll be using nonstop after this class, but for now they are a whole different beast that you will have to deal with later.

Escape Character / Following Style

- \ is the escape "operator"
- bypasses the next character for some functionality

Escape Character / Following Style

- \ is the escape "operator"
- bypasses the next character for some functionality

In [13]:
```
# instead of n being printed, using \ changed it to "newline character'
print('a \nb')
```

```
a
b
```

If I have a line of code that is way too long, I can use the escape character to retain functionality while reducing making it more readable

In [28]:
```
import math
x,y,z = 1,2,3
round(abs(((x ** 2) + (2 * y) - (3 * z)) / (math.sqrt(x) + math.exp(y)
```

Out[28]:
```
0.12
```

In [29]:
```
round(abs(((x ** 2) + (2 * y) - (3 * z)) / \
          (math.sqrt(x) + math.exp(y) * math.log(z)) * \
          (math.sin(x) + math.cos(y) + math.tan(z))), 2)
```

Out[29]:
```
0.12
```

doctest output lengths

In [1]:
```
def some_function(x):
    """
    >>> some_function('marina langlois')
    'marina langloismarina langloismarina langloismarina langloismarina
    """
    return x*5
```

In [ ]:
```
def some_function(x):
    """
    >>> some_function('marina langlois')
    'marina langloismarina langloismarina \
    langloismarina langloismarina langlois'
    """
    return x*5
```

In [ ]:
```
# The correct one!
def some_function(x):
    """
    >>> some_function('marina langlois')
    'marina langloismarina langloismarina \
langloismarina langloismarina langlois'
    """
    return x*5
```

Content

Loops

Loops are used to **repeat computations** many times.

- Two types of loops:
    - While loop: Uses logical conditions, useful for when the number of iterations is unknown (as long as a condition is true, code will run).
    - For loop : Uses an iterable object (ex. list), usually for when the number repetition is known.

In [ ]:
```
while x is True: # watch out for infinite loop!
    # do something
for value in x:
    # do something
```

Lists, Tuples, Sets

List

- Mutable vector of values
- Can store any data type, multiple types at a time
- Elements are acccessed via indexing

Tuple

- **Immutable** vector of values
- all else equal to list

Set

- Mutable vector of values
- Only stores unique elements (removes duplicates)
- otherwise, list like behavior

In [3]:
```
to_5_list = [1,2,3,4,5]
to_5_tup = (1,2,3,4,5)
to_5_set = {1,2,3,4,5}
```

Mutability

- Object is mutable if it can be directly changed after it is created
- If it can't, it is immutable
- Lists are **mutable**
- strings, tuples, and numbers are **immutable**

In [1]:
```
test_str = 'DSC20'
test_str[-2] = '3'
test_str
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent
call last)
Cell In[1], line 2
      1 test_str = 'DSC20'
----> 2 test_str[-2] = '3'
      3 test_str

TypeError: 'str' object does not support item assignment
```

In [2]:
```
test_lst = ['D','S','C','2','0']
test_lst[-2] = '3'
test_lst
```

Out[2]:
```
['D', 'S', 'C', '3', '0']
```

Indexing/Slicing

Indexing/slicing refers to accessing specific element(s) from an iterable object. Two of the most common cases for this are lists and strings. Indexing results in a copy (unless reasssigned)!

- iterable[start:stop:skip] (start:inclusive, stop: NOT inclusive)
- not every section needs to be specified (can just use start or stop or skip)
- sub indexes can be applied (ex. lst[0][0] -> takes the first element of the first element)
- Trying to access an index that doesn't exist in the list will result in an error

In [53]:
```
lst = list(range(2,13))
print("original list: " + str(lst))
print("reversed list: " + str(lst[::-1]))
print("the 2nd to 4th element: " + str(lst[2:4]))
print("every third element from the 1st to 10th element: " + str(lst[1:
print("If I try to slice outside of range: " + str(lst[-100:100]))
```

```
original list: [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
reversed list: [12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2]
the 2nd to 4th element: [4, 5]
every third element from the 1st to 10th element: [3, 6, 9]
If I try to slice outside of range: [2, 3, 4, 5, 6, 7, 8, 9, 10,
11, 12]
```

in-place operations

Before we start exploring functions, it's important to understand what in-place operations are.

Definition: an operation is in-place if the result occurs directly on the original object, rather than a copy. Many in-place functions return None for an output.

What does this actually mean? The result of a not in-place function is a copy, the original object is not modified and the result has to be assigned to a variable to be retained. In-place functions modify the actual object passed in.

In [25]:
```
lst = [1,2,3,4,5]
print('This result is temporary - unless I reassign \
lst to it, lst is not modified: ' + str(lst + [6]))
print("lst's current state: " + str(lst))
lst_new = lst + [6] # reassigning the result retains the output
print("the reassigned lst -> lst_new" + str(lst_new))
```

```
This result is temporary - unless I reassign lst to it, lst is n
ot modified: [1, 2, 3, 4, 5, 6]
lst's current state: [1, 2, 3, 4, 5]
the reassigned lst -> lst_new[1, 2, 3, 4, 5, 6]
```

In [21]:
```
lst = [1,2,3,4,5] # compare that to .append
print("The result of .append() is: " + str(lst.append(6))) # .append()
print("but we can see that the original lst is modified: " + str(lst))
```

```
The result of .append() is: None
but we can see that the original lst is modified: [1, 2, 3, 4,
5, 6]
```

Broad Methods

- iterable.index(element, start, end) -> returns the index of the iterable where the argument is located, otherwise an error is raised.
- iterable.count(element) -> returns the number of times element occurs in the iterable.

In [5]:
```
lst = [5,4,3,2,1,0,0]
print(lst.index(2))
print(lst.count(0))
```

```
3
2
```

List Methods

- list.pop(index) -> removes the element at index. in-place operation; returns the value removed
- list.append(element) -> adds the element to the end of the list. in-place operation
- list.sort() -> sorts the list (as name implies). Can specify ascending or descending
- list.insert(index, element) -> inserts the element at index. in-place operation

In [49]:

```
lst = [5,4,3,2,1,0,0]
print("result of pop first element: " + str(lst.pop(0)))
lst.append(19)
print("after appending 19: "+ str(lst))
lst.sort()
print("after sorting list: " + str(lst))
lst.insert(2, 21)
print("after inserting 21 to index 2" + str(lst))
```

```
result of pop first element: 5
after appending 19: [4, 3, 2, 1, 0, 0, 19]
after sorting list: [0, 0, 1, 2, 3, 4, 19]
after inserting 21 to index 2[0, 0, 21, 1, 2, 3, 4, 19]
```

Checkpoint

Assume the following code has been ran:

In [5]:
```
lst = [10, [12,13,11,10], (2,3,10,4), [4,6, 10, [10,11]], 1]
```

Which statement will extract 10 from the list? Select any that apply.

A. lst[-1]

B. lst[1][-1]

C. lst[2][3]

D. lst[-2][-1][0]

Checkpoint Solution

In [6]:
```
print(lst[-1])
print(lst[1][-1])
print(lst[2][3])
print(lst[-2][-1][0])
```

```
1
10
4
10
```

String Methods

- string.split(separator) -> splits a string into a list of elements on each separator
- "[string]".join(iterable) -> returns the iterable "joined" with the [string] in between each element
- string.lower()/string.upper() -> lowers/uppers all elements in a string
- string.strip() -> removes whitespace from beginning and end of the string
- string.format() -> method to format strings in sections

In [16]:
```
string = "Marina Langlois"
print(string.split())
print('-'.join(string.split()))
print(string.upper())
print("{} is the professor for {}.".format("marina", "DSC20"))
```

```
['Marina', 'Langlois']
Marina-Langlois
MARINA LANGLOIS
marina is the professor for DSC20.
```

Checkpoint

Assume the following code has been ran:

In [4]:
```
statement = "Marina Langlois is the best DSC20 professor ever."
temp = ' '.join(statement.split())
```

Is the following statement True or False? temp is equal to statement

A. True

B. False

Checkpoint Solution

In [5]:
```
temp
```

Out[5]:
```
'Marina Langlois is the best DSC20 professor ever.'
```

In [6]:
```
statement
```

Out[6]:
```
'Marina Langlois is the best DSC20 professor ever.'
```

In [7]:
```
temp == statement
```

Out[7]:
```
True
```

practice questions

Time to do some practice questions! Take about 10-15 minutes to work on the questions.
Feel free to flag me down if you need help/clarification.

Make sure to handwrite! This is practice for your own sake.

YOU MAY KEEP YOUR WORKSHEETS!!

practice question solutions

In [42]:

```
def power(x, a):
    """
    Write a function that calculates x^a.
    Assume both x and a are positive numbers.
    You may not use the in-built power operator.

    >>> power(2, 4)
    16
    >>> power(9,2)
    81
    """
    output = 1
    while a > 0:
        output *= x
        a-=1
    return output
print(power(2, 4))
print(power(9,2))
```

```
16
81
```

In [44]:

```python
# for loop solution
def power(x,a):
    output = 1
    for _ in range(a): # can use i instead of _
        output *= x
    return output
print(power(2, 4))
print(power(9,2))
```

```
16
81
```

In [46]:

```
def yield_even_palindromes(lst):
    """
    Write a function that returns the strings at even indices
    that are also palindromes. Palindromes are words that are
    the same spelled backwards.

    >>> yield_even_palindromes(['121', '232', '01', '443'])
    ['121']
    >>> yield_even_palindromes(['racecar', '0', '0', '1'])
    ['racecar', '0']
    """
    output = []
    even_indexed = lst[::2]
    for val in even_indexed:
        if val == val[::-1]:
            output.append(val)
    return output
print(yield_even_palindromes(['121', '232', '01', '443']))
print(yield_even_palindromes(['racecar', '0', '0', '1']))
```

```
['121']
['racecar', '0']
```

In [47]:

```
def strictly_increasing(lst):
    """
    Write a function that takes in a list of integers and
    returns a new list containing only the numbers that are
    in increasing order. Numbers should appear in the same
    order in the input and output list.

    >>> increasing([1, 3, 2, 4, 5, 8, 7, 6, 9])
    [1, 3, 4, 5, 8, 9]
    """
    output = []
    for num in lst:
        if len(output) == 0:
            output.append(num)
        elif num > output[-1]:
            output.append(num)
    return output
print(strictly_increasing([1, 3, 2, 4, 5, 8, 7, 6, 9]))
```

```
[1, 3, 4, 5, 8, 9]
```

In [50]:

```
def process_string(word):
    """
    Write a function that takes in a string and returns
    the first 3 characters if its length is odd and
    decomposes the string as a list of characters otherwise.

    >>> process_string('abcde')
    'abc'
    >>> process_string('nicole')
    ['n', 'i', 'c', 'o', 'l', 'e']
    """
    if len(word)%2==1:
        return word[:3]
    else:
        output = []
        for c in word:
            output.append(c)
        return output

print(process_string('abcde'))
print(process_string('nicole'))
```

```
abc
['n', 'i', 'c', 'o', 'l', 'e']
```

In [51]:

```python
# alternative
def process_string(word):
    """
    Write a function that takes in a string and returns
    the first 3 characters if its length is odd and
    decomposes the string as a list of characters otherwise.

    >>> process_string('abcde')
    'abc'
    >>> process_string('nicole')
    ['n', 'i', 'c', 'o', 'l', 'e']
    """
    if len(word)%2==1:
        return word[:3]
    else:
        return list(word)

print(process_string('abcde'))
print(process_string('nicole'))
```

```
abc
['n', 'i', 'c', 'o', 'l', 'e']
```

How many of the questions could you complete?

A. 0

B. 1

C. 2

D. 3

E. 4

Thanks for coming!