Discussion 2

DSC 20, Spring 2023

Debugging

Agenda

- list, dictionary, doctests overview
- debugging practices
- reading and finding errors in code
- python tutor debugging + worksheet

Lists

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

Checkpoint

Write a function that returns a list, where each name in the ***names*** list is replaced with the string "Even" if the name has even length, or "Odd" otherwise.

If the ***names*** list is empty, return a list with the string "Empty list was given" in it.

Checkpoint

Write a function that returns a list, where each name in the ***names*** list is replaced with the string "Even" if the name has even length, or "Odd" otherwise.
If the ***names*** list is empty, return a list with the string "Empty list was given" in it.

In [1]:
```
def odd_even_list(names):
    '''
    Function that checks the length parity of a list of names.

    Args:
        names (list): list of strings to be considered

    Returns:
        a list consisting of "Odd" or "Even" where each element indicat
        length parity of each string in names. If names is empty, retur
        a list with the string "Empty list was given".

    >>> odd_even_list(["Marina", "Michelle", "James", "Darren"])
    ['Even', 'Even', 'Odd', 'Even']
    >>> odd_even_list([])
    ['Empty list was given']
    '''
    # Write your implementation here
    return
```

Checkpoint Solution

Checkpoint Solution

In [2]:
```
def odd_even_list(names):
    '''
    Function that checks the length parity of a list of names.

    Args:
        names (list): list of strings to be considered

    Returns:
        a list consisting of "Odd" or "Even" where each element indicat
        length parity of each string in names. If names is empty, retur
        a list with the string "Empty list was given".

    >>> odd_even_list(["Marina", "Michelle", "James", "Darren"])
    ['Even', 'Even', 'Odd', 'Even']
    >>> odd_even_list([])
    ['Empty list was given']
    '''
    if not len(names): # check for empty list
        return ['Empty list was given']
    else:
        parity_list = []
        for name in names:
            if len(name)%2 == 0: #check for even length
                parity_list.append('Even')
                # alternate solution: parity_list+=['Even']
            else: #since only two cases are possible, no need to check
                parity_list.append('Odd')
```

```
            # alternate solution: parity_list+=['Odd']
    return parity_list
```

Checkpoint Solution

In [2]:
```
def odd_even_list(names):
    '''
    Function that checks the length parity of a list of names.

    Args:
        names (list): list of strings to be considered

    Returns:
        a list consisting of "Odd" or "Even" where each element indicat
        length parity of each string in names. If names is empty, retur
        a list with the string "Empty list was given".

    >>> odd_even_list(["Marina", "Michelle", "James", "Darren"])
    ['Even', 'Even', 'Odd', 'Even']
    >>> odd_even_list([])
    ['Empty list was given']
    '''
    if not len(names): # check for empty list
        return ['Empty list was given']
    else:
        parity_list = []
        for name in names:
            if len(name)%2 == 0: #check for even length
                parity_list.append('Even')
                # alternate solution: parity_list+=['Even']
            else: #since only two cases are possible, no need to check
                parity_list.append('Odd')
```

```
                # alternate solution: parity_list+=['Odd']
    return parity_list
```

In [3]:
```
odd_even_list(["Marina", "Michelle", "James", "Darren"])
```

Out[3]:
```
['Even', 'Even', 'Odd', 'Even']
```

Dictionaries

- Mutable storage of key, value pairs
- Can store any data type, multiple at a time
- Elements are acccessed via keys
- keys must be **hashable** and **unique**

note: hashablility correlates to the stability of the data - essentially, **data that can't change is hashable** (int, str, tuple, etc.) while **data that can change is not hashable (list, dictionary)**

Checkpoint

Write a function that converts a list of tuples into a dictionary, where the first and second elements of each tuple are the key, value for the dictionary respectively.

Assume that each tuple will have exactly 2 elements.

Checkpoint

Write a function that converts a list of tuples into a dictionary, where the first and second elements of each tuple are the key, value for the dictionary respectively.
Assume that each tuple will have exactly 2 elements.

In [4]:
```
def tuple_to_dictionary(tup_list):
    '''
    Function that converts a list of 2-element tuples into a dictionary

    Args:
        tup_list (list): list of 2-element tuples to be considered

    Returns:
        a dictionary where the keys are the first elements and the valu

    >>> tuple_to_dictionary([('Ben', ['Badminton', 'Track','X-country']
    {'Ben':['Badminton', 'Track', 'X-country'], 'Charisse':['Tennis'],
    >>> tuple_to_dictionary([])
    {}
    '''
    # Write your implementation here
    return
```

Checkpoint Solution

Checkpoint Solution

In [5]:
```
def tuple_to_dictionary(tup_list):
    '''
    Function that converts a list of 2-element tuples into a dictionary

    Args:
        tup_list (list): list of 2-element tuples to be considered

    Returns:
        a dictionary where the keys are the first elements and the valu

    >>> tuple_to_dictionary([('Ben', ['Badminton', 'Track','X-country']
    {'Ben':['Badminton', 'Track', 'X-country'], 'Charisse':['Tennis'],
    >>> tuple_to_dictionary([])
    {}
    '''
    converted_result = {}
    for key, value in tup_list: # since 2-element tuples are guarantee
        converted_result[key] = value
        #alternative solution:
        # for tup in tup_list:
        #     converted_result[tup[0]] = tup[1]
    return converted_result
```

Checkpoint Solution

In [5]:
```
def tuple_to_dictionary(tup_list):
    '''
    Function that converts a list of 2-element tuples into a dictionary

    Args:
        tup_list (list): list of 2-element tuples to be considered

    Returns:
        a dictionary where the keys are the first elements and the valu

    >>> tuple_to_dictionary([('Ben', ['Badminton', 'Track','X-country']
    {'Ben':['Badminton', 'Track', 'X-country'], 'Charisse':['Tennis'],
    >>> tuple_to_dictionary([])
    {}
    '''
    converted_result = {}
    for key, value in tup_list: # since 2-element tuples are guarantee
        converted_result[key] = value
        #alternative solution:
        # for tup in tup_list:
        #     converted_result[tup[0]] = tup[1]
    return converted_result
```

In [6]:
```
tuple_to_dictionary([('Ben', ['Badminton', 'Track','X-country']),('Char
```

Out[6]:

```
{'Ben': ['Badminton', 'Track', 'X-country'],
 'Charisse': ['Tennis'],
 'Nicole': ['Dance', 'Golf']}
```

Doctests

- Tests to check that your function works as intended
- denoted by the '>>> ' symbol (space included)!
- the line right after the '>>> ' represents the intended output
- well written doctests make sure your code is logically sound in all cases

note: if you want to see the doctests you passed when you run them, add a '-v' at the end.

Doctests

- Tests to check that your function works as intended
- denoted by the '>>> ' symbol (space included)!
- the line right after the '>>> ' represents the intended output
- well written doctests make sure your code is logically sound in all cases

note: if you want to see the doctests you passed when you run them, add a '-v' at the end.

```
~/De/tutor/DSC20_sp_2023/hw/hw01 / main !1
python3 -m doctest hw01_sol.py
```

Doctests

- Tests to check that your function works as intended
- denoted by the '>>> ' symbol (space included)!
- the line right after the '>>> ' represents the intended output
- well written doctests make sure your code is logically sound in all cases

note: if you want to see the doctests you passed when you run them, add a '-v' at the end.

```
~/De/tutor/DSC20_sp_2023/hw/hw01 / main !1
python3 -m doctest hw01_sol.py
```

```
~/De/tutor/DSC20_sp_2023/hw/hw01 / main !1
python3 -m doctest hw01_sol.py -v
```

Doctests

- Tests to check that your function works as intended
- denoted by the '>>> ' symbol (space included)!
- the line right after the '>>> ' represents the intended output
- well written doctests make sure your code is logically sound in all cases

note: if you want to see the doctests you passed when you run them, add a '-v' at the end.

```
~/De/tutor/DSC20_sp_2023/hw/hw01 / main !1
python3 -m doctest hw01_sol.py
```

```
~/De/tutor/DSC20_sp_2023/hw/hw01 / main !1
python3 -m doctest hw01_sol.py -v
```

```
1 items had no tests:
    hw01_sol
11 items passed all tests:
   4 tests in hw01_sol.age_average
   3 tests in hw01_sol.all_distances
   5 tests in hw01_sol.close_to_25
   3 tests in hw01_sol.helper_distance
   3 tests in hw01_sol.main_driver
   2 tests in hw01_sol.message
   3 tests in hw01_sol.name_scramble
   3 tests in hw01_sol.places_names
   3 tests in hw01_sol.seat_number
   6 tests in hw01_sol.split_teams
   3 tests in hw01_sol.suv_vs_minivan
38 tests in 12 items.
38 passed and 0 failed.
Test passed.
```

Errors

- Errors are a special **python class**
- Encountered errors immediately exit code (short-circuit)
- There's 2 types of broad errors: **Syntax error** and **Runtime error**
- Errors are triggered when improper code is attempted

for syntax error: python syntax is not followed (code will not run at all)

for runtime error: written code in the file does not run

Errors

- Errors are a special **python class**
- Encountered errors immediately exit code (short-circuit)
- There's 2 types of broad errors: **Syntax error** and **Runtime error**
- Errors are triggered when improper code is attempted

  for syntax error: python syntax is not followed (code will not run at all)

  for runtime error: written code in the file does not run

Common Errors Encountered

- **KeyError** - related to dictionaries; attempted access using a key not present in the object
- **IndexError** - related to lists/strings; attempted acccess of an index that's out of range
- **TypeError** - attempt to unify non - matching data types (ex. str + int) or attempt to access unknown attribute of datatype
- **FileNotFoundError** - related to files; attempted to open a file name that can't be found

Checkpoint - Question 1

Assume the following line of code is ran:

Checkpoint - Question 1

Assume the following line of code is ran:

In [7]:
```
sample_dict = {'Professor': ['Marina'], 'Tutors': ['Ben', 'Charisse'],
```

Checkpoint - Question 1

Assume the following line of code is ran:

In [7]:
```
sample_dict = {'Professor': ['Marina'], 'Tutors': ['Ben', 'Charisse'],
```

We want to get 'Professor' out from the dictionary, so we write the following code

Checkpoint - Question 1

Assume the following line of code is ran:

In [7]:
```
sample_dict = {'Professor': ['Marina'], 'Tutors': ['Ben', 'Charisse'],
```

We want to get 'Professor' out from the dictionary, so we write the following code

In [ ]:
```
sample_dict.keys()[0]
```

Checkpoint - Question 1

Assume the following line of code is ran:

In [7]:
```
sample_dict = {'Professor': ['Marina'], 'Tutors': ['Ben', 'Charisse'],
```

We want to get 'Professor' out from the dictionary, so we write the following code

In [ ]:
```
sample_dict.keys()[0]
```

What is the result? Is there an error? If so, what error (and why?)

Checkpoint - Question 1

Assume the following line of code is ran:

In [7]:
```
sample_dict = {'Professor': ['Marina'], 'Tutors': ['Ben', 'Charisse'],
```

We want to get 'Professor' out from the dictionary, so we write the following code

In [ ]:
```
sample_dict.keys()[0]
```

What is the result? Is there an error? If so, what error (and why?)

In [8]:
```
sample_dict.keys()[0]
```

```
---------------------------------------------------------------
-----------
TypeError                                 Traceback (most recent
call last)
/var/folders/s3/jrstqxb973db3gqjptn3m3nr0000gp/T/ipykernel_2033/
2302762907.py in <module>
----> 1 sample_dict.keys()[0]

TypeError: 'dict_keys' object is not subscriptable
```

Checkpoint - Question 1

Assume the following line of code is ran:

In [7]:
```
sample_dict = {'Professor': ['Marina'], 'Tutors': ['Ben', 'Charisse'],
```

We want to get 'Professor' out from the dictionary, so we write the following code

In [ ]:
```
sample_dict.keys()[0]
```

What is the result? Is there an error? If so, what error (and why?)

In [8]:
```
sample_dict.keys()[0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/s3/jrstqxb973db3gqjptn3m3nr0000gp/T/ipykernel_2033/2302762907.py in <module>
----> 1 sample_dict.keys()[0]

TypeError: 'dict_keys' object is not subscriptable
```

the result of .keys() is not a list, you would have to cast it into a list for this to work properly

Checkpoint - Question 1

Assume the following line of code is ran:

In [7]:
```
sample_dict = {'Professor': ['Marina'], 'Tutors': ['Ben', 'Charisse'],
```

We want to get 'Professor' out from the dictionary, so we write the following code

In [ ]:
```
sample_dict.keys()[0]
```

What is the result? Is there an error? If so, what error (and why?)

In [8]:
```
sample_dict.keys()[0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/s3/jrstqxb973db3gqjptn3m3nr0000gp/T/ipykernel_2033/2302762907.py in <module>
----> 1 sample_dict.keys()[0]

TypeError: 'dict_keys' object is not subscriptable
```

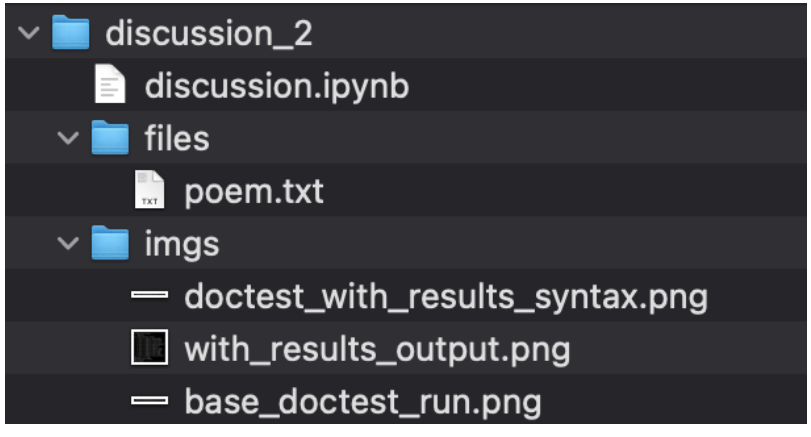
the result of .keys() is not a list, you would have to cast it into a list for this to work properly

In [9]:
```
list(sample_dict.keys())[0]
```

Out[9]:
```
'Professor'
```

Checkpoint - Question 2

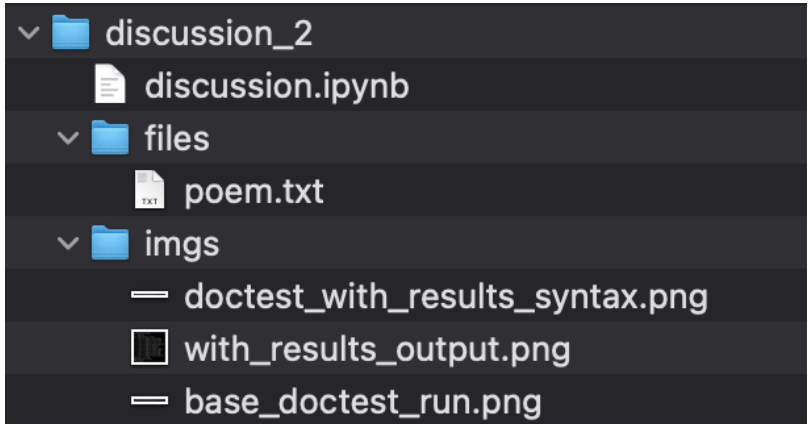
assume the following file structure:



We want to print out the poem, so we write the following code:

Checkpoint - Question 2

assume the following file structure:

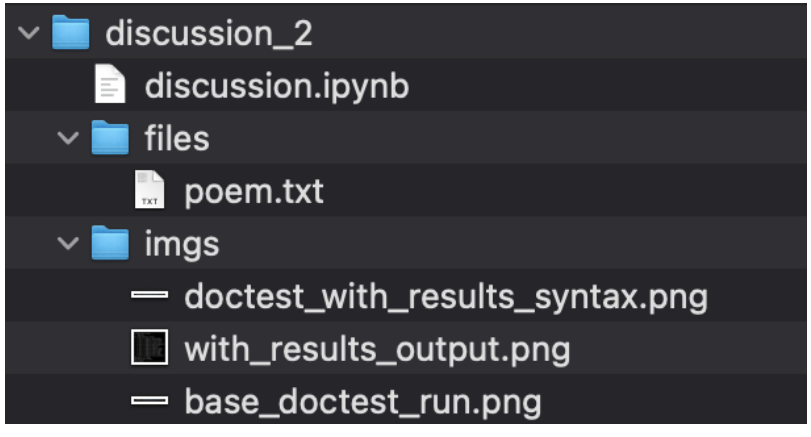


We want to print out the poem, so we write the following code:

In [ ]:
```
with open('poem.txt', 'r') as f:
    print(f.readlines())
```

Checkpoint - Question 2

assume the following file structure:



We want to print out the poem, so we write the following code:

In [ ]:
```
with open('poem.txt', 'r') as f:
    print(f.readlines())
```

What is the result? Is there an error? If so, what error (and why?)

In [10]:
```
with open('poem.txt', 'r') as f:
    print(f.readlines())
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
/var/folders/s3/jrstqxb973db3gqjptn3m3nr0000gp/T/ipykernel_2033/4097085444.py in <module>
----> 1 with open('poem.txt', 'r') as f:
      2     print(f.readlines())

FileNotFoundError: [Errno 2] No such file or directory: 'poem.txt'
```

In [10]:
```
with open('poem.txt', 'r') as f:
    print(f.readlines())
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
/var/folders/s3/jrstqxb973db3gqjptn3m3nr0000gp/T/ipykernel_2033/4097085444.py in <module>
----> 1 with open('poem.txt', 'r') as f:
      2     print(f.readlines())

FileNotFoundError: [Errno 2] No such file or directory: 'poem.txt'
```

the file name is not correct, we are missing the parent folder for poem.txt (files/)

In [11]:

```python
with open('files/poem.txt', 'r') as f:
    for line in f.readlines():
        print(line)
    print()
    print("by E.E. Cummings")
```

```
l(a

le

af

fa

ll

s)

one

l

Iness

by E.E. Cummings
```

Checkpoint - Question 3

Look familiar? This code generates an error, figure out what line and why the error occurs.

Checkpoint - Question 3

Look familiar? This code generates an error, figure out what line and why the error occurs.

In [12]:
```
def odd_even_list(names):
    '''
    Function that checks the length parity of a list of names.

    Args:
        names (list): list of strings to be considered

    Returns:
        a list consisting of "Odd" or "Even" where each element indicat
        length parity of each string in names. If names is empty, retur
        a list with the string "Empty list was given".

    >>> odd_even_list(["Marina", "Michelle", "James", "Darren"])
    ['Even', 'Even', 'Odd', 'Even']
    >>> odd_even_list([])
    ['Empty list was given']
    '''
    if not len(names): # check for empty list
        return ['Empty list was given']
    else:
        parity_list = []
        for name in names:
            if len(name)%2 == 0: #check for even length
                parity_list = parity_list.append('Even')
            else: #since only two cases are possible, no need to check
```

```
                parity_list.append('Odd')
    return parity_list
```

In [13]:
```
odd_even_list(["Marina", "Michelle", "James", "Darren"])
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/var/folders/s3/jrstqxb973db3gqjptn3m3nr0000gp/T/ipykernel_2033/901610094.py in <module>
----> 1 odd_even_list(["Marina", "Michelle", "James", "Darren"])

/var/folders/s3/jrstqxb973db3gqjptn3m3nr0000gp/T/ipykernel_2033/2585500232.py in odd_even_list(names)
     22         for name in names:
     23             if len(name)%2 == 0: #check for even length
---> 24                 parity_list = parity_list.append('Even')
     25             else: #since only two cases are possible, no need to check odd
     26                 parity_list.append('Odd')

AttributeError: 'NoneType' object has no attribute 'append'
```

In [13]:
```
odd_even_list(["Marina", "Michelle", "James", "Darren"])
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/var/folders/s3/jrstqxb973db3gqjptn3m3nr0000gp/T/ipykernel_2033/901610094.py in <module>
----> 1 odd_even_list(["Marina", "Michelle", "James", "Darren"])

/var/folders/s3/jrstqxb973db3gqjptn3m3nr0000gp/T/ipykernel_2033/2585500232.py in odd_even_list(names)
     22         for name in names:
     23             if len(name)%2 == 0: #check for even length
---> 24                 parity_list = parity_list.append('Even')
     25             else: #since only two cases are possible, no need to check odd
     26                 parity_list.append('Odd')

AttributeError: 'NoneType' object has no attribute 'append'
```

Answer: Line 24, AttributeError (NoneType)

Why?: .append() is an in-place function. This means that its operation occurred on the original list rather than a copy, returning nothing. This means that during our code, on line 24, our parity_list variable was accidently reassigned to be equal to the result of .append(), which is None. In the next loop, the same operation is attempted on None instead of a list.

note: functions without a return evaluate to None.

For Reference - Original Solution

For Reference - Original Solution

In [14]:

```
def odd_even_list(names):
    '''
    Function that checks the length parity of a list of names.

    Args:
        names (list): list of strings to be considered

    Returns:
        a list consisting of "Odd" or "Even" where each element indicat
        length parity of each string in names. If names is empty, retur
        a list with the string "Empty list was given".

    >>> odd_even_list(["Marina", "Michelle", "James", "Darren"])
    ['Even', 'Even', 'Odd', 'Even']
    >>> odd_even_list([])
    ['Empty list was given']
    '''
    if not len(names): # check for empty list
        return ['Empty list was given']
    else:
        parity_list = []
        for name in names:
            if len(name)%2 == 0: #check for even length
                parity_list.append('Even')
                # alternate solution: parity_list+=['Even']
            else: #since only two cases are possible, no need to check
                parity_list.append('Odd')
```

```
            # alternate solution: parity_list+=['Odd']
    return parity_list
```

Debugging Practices

- **Read the error**: error type and line it occurs
- Based on the error type, try to see what part of the code is causing it
- Make sure to **print things** out to try and see where your code breaks!
- If you're not sure, use **Google/stackoverflow** to look at common causes
- If you still can't figure it out, plug into **pythontutor**

Debugging + Python Tutor

live demo

original file is linked here

Worksheet Time!

feel free to discuss or work with the people around you, the point is to get used to writing code on paper

Worksheet Review

Question 1

Question 1

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.

Question 1

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.

In [15]:
```
def increasing(lst):
    """
    >>> increasing([1, 3, 2, 4, 5, 8, 7, 6, 9])
    [1, 3, 4, 5, 8, 9]
    """
    # Write your implementation here
    return
```

Solution

Solution

In [16]:

```
def increasing(lst):
    """
    >>> 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
```

Solution

In [16]:
```
def increasing(lst):
    """
    >>> 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
```

In [17]:
```
increasing([1, 3, 2, 4, 5, 8, 7, 6, 9])
```

Out[17]:
```
[1, 3, 4, 5, 8, 9]
```

Question 2

Question 2

Suppose you are given a dictionary that contains information about students and their grades for different classes. Write a function that takes in such a dictionary and returns a new dictionary that contains the average grade for each student. The output dictionary should have student names as keys and their average grades as values.

Question 2

Suppose you are given a dictionary that contains information about students and their grades for different classes. Write a function that takes in such a dictionary and returns a new dictionary that contains the average grade for each student. The output dictionary should have student names as keys and their average grades as values.

In [ ]:

```
def avg_grade(grades):
    """
    >>> avg_grade({"Alice": {"math": 85, "dsc": 90, "english": 80},
     "Bob": {"math": 92, "dsc": 88, "english": 95}})
    {"Alice": 85.0, "Bob": 91.67}
    """
    # Write your implementation here
    return
```

Solution

Solution

In [18]:

```
def avg_grade(grades):
    """
    >>> avg_grade({"Alice": {"math": 85, "dsc": 90, "english": 80},
     "Bob": {"math": 92, "dsc": 88, "english": 95}})
    {"Alice": 85.0, "Bob": 91.67}
    """
    avg_grades = {}
    for name, classes in grades.items():
        grades = classes.values()
        avg_grades[name] = round((sum(grades) / len(grades)), 2)
    return avg_grades
```

Solution

In [18]:
```
def avg_grade(grades):
    """
    >>> avg_grade({"Alice": {"math": 85, "dsc": 90, "english": 80},
     "Bob": {"math": 92, "dsc": 88, "english": 95}})
    {"Alice": 85.0, "Bob": 91.67}
    """
    avg_grades = {}
    for name, classes in grades.items():
        grades = classes.values()
        avg_grades[name] = round((sum(grades) / len(grades)), 2)
    return avg_grades
```

In [19]:
```
avg_grade({"Alice": {"math": 85, "dsc": 90, "english": 80},
           "Bob": {"math": 92, "dsc": 88, "english": 95}})
```

Out[19]:
```
{'Alice': 85.0, 'Bob': 91.67}
```

Question 3

Question 3

You go grocery shopping and note down the name, quantity, and price per unit of each item you buy in the variable shopping. What does the following code do and what will be printed after running it?

Question 3

You go grocery shopping and note down the name, quantity, and price per unit of each item you buy in the variable shopping. What does the following code do and what will be printed after running it?

In [ ]:

```
shopping = [('banana', 5, 0.75),
            ('avocado', 4, 1.5),
            ('soda', 8, 0.5),
            ('peach', 10, 1.70)]

total = 0
item_cost = {}
for item in shopping:
    name = item[0]
    quantity = int(item[1])
    price = float(item[2])
    cost = quantity * price
    item_cost[name] = cost
    total += cost

print(item_cost)
print(total)
```

Solution

Solution

What does the code do? For each item bought, it is stored in a dictionary, with the name as the key and the total cost of the item as the value. It also calculates the total cost of everything bought.

Solution

What does the code do? For each item bought, it is stored in a dictionary, with the name as the key and the total cost of the item as the value. It also calculates the total cost of everything bought.

In [20]:

```
shopping = [('banana', 5, 0.75),
            ('avocado', 4, 1.5),
            ('soda', 8, 0.5),
            ('peach', 10, 1.70)]

total = 0
item_cost = {}
for item in shopping:
    name = item[0]
    quantity = int(item[1])
    price = float(item[2])
    cost = quantity * price
    item_cost[name] = cost
    total += cost

print(item_cost)
print(total)
```

```
{'banana': 3.75, 'avocado': 6.0, 'soda': 4.0, 'peach': 17.0}
30.75
```

Question 4

Question 4

For each of the following sections, name the error type (KeyError, IndexError, SyntaxError, TypeError):

Question 4

For each of the following sections, name the error type (KeyError, IndexError, SyntaxError, TypeError):

In [ ]:
```
def send_email(to, message):
    tutors = {'Ben': 'bhc001@ucsd.edu', 'Charisse': 'chao@ucsd.edu',
     'Nicole': 'nwzhang@ucsd.edu', 'Jessica': 'yuhung@ucsd.edu'}
    return "Email sent to " + tutors[to] + ": " + message

send_email("Marina", "There's too many types of errors")
```

Question 4

For each of the following sections, name the error type (KeyError, IndexError, SyntaxError, TypeError):

In [ ]:
```
def send_email(to, message):
    tutors = {'Ben': 'bhc001@ucsd.edu', 'Charisse': 'chao@ucsd.edu',
     'Nicole': 'nwzhang@ucsd.edu', 'Jessica': 'yuhung@ucsd.edu'}
    return "Email sent to " + tutors[to] + ": " + message

send_email("Marina", "There's too many types of errors")
```

In [ ]:
```
def send_email(info):
    tutors = {'Ben': 'bhc001@ucsd.edu', 'Charisse': 'chao@ucsd.edu',
     'Nicole': 'nwzhang@ucsd.edu', 'Jessica': 'yuhung@ucsd.edu'}

    recipient = info[0]
    message = info[1]
    return "Email sent to " + recipient + ": " + message

send_email(["There's too many types of errors"])
```

Solution

Solution

In [21]:
```
def send_email(to, message):
    tutors = {'Ben': 'bhc001@ucsd.edu', 'Charisse': 'chao@/ucsd.edu',
     'Nicole': 'nwzhang@ucsd.edu', 'Jessica': 'yuhung@ucsd.edu'}
    return "Email sent to " + tutors[to] + ": " + message
```

Solution

In [21]:
```
def send_email(to, message):
    tutors = {'Ben': 'bhc001@ucsd.edu', 'Charisse': 'chao@/ucsd.edu',
     'Nicole': 'nwzhang@ucsd.edu', 'Jessica': 'yuhung@ucsd.edu'}
    return "Email sent to " + tutors[to] + ": " + message
```

In [22]:
```
send_email("Marina", "There's too many types of errors")
```

```
---------------------------------------------------------------
-----------
KeyError                                  Traceback (most recent
call last)
/var/folders/s3/jrstqxb973db3gqjptn3m3nr0000gp/T/ipykernel_2033/
2571869973.py in <module>
----> 1 send_email("Marina", "There's too many types of errors")

/var/folders/s3/jrstqxb973db3gqjptn3m3nr0000gp/T/ipykernel_2033/
1637858142.py in send_email(to, message)
      2     tutors = {'Ben': 'bhc001@ucsd.edu', 'Charisse': 'cha
o@/ucsd.edu',
      3      'Nicole': 'nwzhang@ucsd.edu', 'Jessica': 'yuhung@uc
sd.edu'}
----> 4     return "Email sent to " + tutors[to] + ": " + messag
e

KeyError: 'Marina'
```

Solution

Solution

In [23]:
```
def send_email(info):
    tutors = {'Ben': 'bhc001@ucsd.edu', 'Charisse': 'chao@ucsd.edu',
     'Nicole': 'nwzhang@ucsd.edu', 'Jessica': 'yuhung@ucsd.edu'}

    recipient = info[0]
    message = info[1]
    return "Email sent to " + recipient + ": " + message
```

Solution

In [23]:
```
def send_email(info):
    tutors = {'Ben': 'bhc001@ucsd.edu', 'Charisse': 'chao@ucsd.edu',
     'Nicole': 'nwzhang@ucsd.edu', 'Jessica': 'yuhung@ucsd.edu'}

    recipient = info[0]
    message = info[1]
    return "Email sent to " + recipient + ": " + message
```

In [24]:
```
send_email(["There's too many types of errors"])
```

```
---------------------------------------------------------------
-----------
IndexError                                Traceback (most recent
call last)
/var/folders/s3/jrstqxb973db3gqjptn3m3nr0000gp/T/ipykernel_2033/
2536149692.py in <module>
----> 1 send_email(["There's too many types of errors"])

/var/folders/s3/jrstqxb973db3gqjptn3m3nr0000gp/T/ipykernel_2033/
1660277811.py in send_email(info)
      4
      5     recipient = info[0]
----> 6     message = info[1]
      7     return "Email sent to " + recipient + ": " + message

IndexError: list index out of range
```

Thanks for Coming!

Don't forget discussion quiz on Canvas

Next week, we will be running through questions from old exams in preparation for midterm 1