CSC108H Winter 2024 Worksheet 20 : Files

Reading from files:

1. We have a spreadsheet file that we've opened and assigned to `f`:

   ```
   f = open('budgie_budget.csv')
   ```

   Consider these code fragments:

   (a)
   ```
   for line in f:
       print(line)
   ```

   (b)
   ```
   line = f.readline()
   for line in f:
       print(line)
   ```

   (c)
   ```
   for line in f:
       print(line)
       f.readline()
   ```

   (d)
   ```
   print(f.readlines()[0])
   ```

   Fill in the blank next to each description below with the code fragment from above, (a), (b), (c) or (d), that it describes.

   (1) prints only the first line _____(d)_____

   (2) prints every line except the first _____(b)_____

   (3) prints all lines _____(a)_____

   (4) prints every second line _____(c)_____

   ```
   from typing import TextIO
   ```

2. Many Unix-like systems (like OSX and the Teaching Labs) have a file of correctly spelled words. On a Mac, the path to the file is `/usr/share/dict/words`. On Teaching Labs, the path to the file is `/etc/dictionaries-common/words`. See below some of those words (the file contains both capitalized and lowercase words); complete the function on the right:

   ```
   Zworykin   |  def is_correct(file: TextIO, word: str) -> bool:
   Zyrtec     |      """Return True if and only if word is in file.
   Zyrtec's   |
   a          |      >>> words_file = open('dictionary.txt')
   aardvark   |      >>> is_correct(words_file, 'Zyrtec')
   aardvarks  |      True
   abaci      |      >>> words_file.close()
   aback      |      >>> words_file = open('dictionary.txt')
              |      >>> is_correct(words_file, 'lolz')
              |      False
              |      >>> words_file.close()
              |      """
                       return any(word == line.strip() for line in file)
   ```

CSC108H Winter 2024 Worksheet 20 : Files

Writing to files:

3. Consider this code:

```
budget_file = open('budgie_budget.txt', 'w')
budget_file.write('Seed: $10/month')
budget_file.write('Cage: $50')
budget_file.close()
```

What will the contents of budgie_budget.txt look like after this code is run?

(a)
```
'Seed: $10/month'
'Cage: $50'
```

(b)
```
Seed: $10/month
Cage: $50
```

(c)
```
Seed: $10/month Cage: $50
```

(d)
```
Seed: $10/monthCage: $50
```

(e)
```
Cage: $50
```

(f)
```
'Seed: $10/month''Cage: $50'
```

4. Complete the following function:

```
def write_ascii_triangle(outfile: TextIO, block: str, sidelength: int) -> None:
    """Write an ascii isosceles right triangle using block that is sidelength
    characters wide and high to outfile. The right angle should be in the
    upper-left corner.

    Precondition: len(block) == 1

    For example, given block="@" and sidelength=4, the
    following should be written to the file:

    @@@@
    @@@
    @@
    @
    """
    for i in range(sidelength, 0, -1):
        outfile.write(block * i + '\n')
    outfile.close()
```

CSC108H Winter 2024 Worksheet 21 : Structured Files

The NHL-data.txt file format:

The file `NHL-data.txt` is a text file that contains the characters:

```
Toronto Maple Leafs\n2\n2\n1\n0\n0\n2\nGrande Prairie Storm\nMontreal Canadiens\n1\n2\n1\n0\n2
```

Interpreting the newline character `\n` as an instruction to move to a new line allows us to visualize the file as:

```
Toronto Maple Leafs
2
2
1
0
0
2
Grande Prairie Storm
Montreal Canadiens
1
2
1
0
2
```

The file contains hockey team names, with each team name followed by the number of points earned in each game in the NHL season so far. Note that the `Toronto Maple Leafs` have played 6 games, the `Montreal Canadiens` have played 5 games and the `Grande Prairie Storm` have not played in the NHL yet. This file is more complicated than the `dictionary.txt` file since the lines can contain either a team name or a number of points, and a line with a number of points belongs to the most recently read team name.

Before writing code that reads a file that does not have a simple structure, it can be helpful to write an abstract description of the file contents. The `NHL-data.txt` looks like:

```
Team Name 1
points from Team Name 1's game 1
points from Team Name 1's game 2
...
points from Team Name 1's game N1
Team Name 2
points from Team Name 2's game 1
...
points from Team Name 2's game N2
Team Name 3
points from Team Name 3's game 1
...
points from Team Name 3's game N3
```

Note that in our example file `N1` is 6, `N2` is 0 (no NHL games yet) and `N3` is 5.

More generally, a `NHL-data.txt` file contains a sequence of team names, with each team name followed by the number of points earned by the team in each NHL game. The number of team names could be 0, 1 or some larger number, and the number of games played by each team could be 0, 1 or some larger number. **We will assume that the team names and number of points earned each have a valid value.**

CSC108H Winter 2024 Worksheet 21 : Structured Files

Writing code to read more complicated files:

The following function is passed a reference to an already opened file that has the same format as the `NHL-data.txt` file. Fill in the boxes below with appropriate `python` code so the function works as described. Before writing code, answer the questions:

1. What value will be read when the end of the file has been reached?
2. How can you determine whether or not a line contains a number of game points?
3. When might it not be possible to determine an average number of points per game?

```
from typing import TextIO

def points_per_game(game_data: TextIO) -> list[list]:
    """Return a list containing the team name and the average number of points
    earned per game for each team in the open file game_data. If the team has
    no games, use None instead of an average number of points.

    >>> input_file = open('NHL-data.txt')
    >>> points_per_game(input_file)
    [['Toronto Maple Leafs', 1.17], ['Grande Prairie Storm', None], \
['Montreal Canadiens', 1.2]]
    >>> input_file.close()
    """
    result = []

    # read a line from the file to set up for the outer while loop condition
    line = game_data.readline().strip()
    while line != '':
        # due to the structure of the file, line contains a team name.

        team_name = [                    ]
        # set up accumulator variables for the team.
        games_played = [          ]

        total_points = [          ]

        # read and process game points until: new team name is read or end of file is reached
        [                                                            ]
        while [                                                  ]:
            total_points = total_points + [                              ]
            games_played = games_played + 1
            [                                                            ]

        # add results for team_name to result list
        if [                                        ]:
            result.append([team_name, round(total_points / games_played, 2)])
        else:
            result.append([team_name, None])
    return result
```

CSC108H Winter 2024 Worksheet 22 : Dictionaries

1. Consider this code:

```
name_to_binomial = {'human': 'Homo sapiens',
                    'dog': 'Canis familiaris',
                    'narwhal': 'Monodon monoceros'}
```

Circle the expressions that evaluate to `True`.

(a) `'dog' in name_to_binomial`

(b) `'Canis familiaris' in name_to_binomial`

(c) `name_to_binomial[0] == 'human'`

(d) `len(name_to_binomial) == 3`

(e)
```
name_to_binomial == {'dog': 'Canis familiaris',
                     'narwhal': 'Monodon monoceros',
                     'human': 'Homo sapiens'}
```

2. Consider this code:

```
animal_to_locomotion = {'fish': ['swim'],
                        'kangaroo': ['hop'],
                        'frog': ['swim', 'hop']}
```

Indicate whether each statement will cause an error and, if not, whether the statement will increase the number of key/value pairs in the dictionary:

Statement	**Error?** (yes or no)	**Increases length of dictionary?** (yes or no)
`animal_to_locomotion['human'] = ['swim', 'run', 'walk', 'airplane']`		
`animal_to_locomotion['orangutan'].append('brachiate')`		
`animal_to_locomotion['kangaroo'].append('airplane')`		
`animal_to_locomotion['frog'] = ['tapdance']`		
`animal_to_locomotion['dolphin'] = animal_to_locomotion['fish']`		

CSC108H Winter 2024 Worksheet 22 : Dictionaries

3. The express checkout is for grocery orders with 8 or fewer items. Complete the examples in the docstring and then complete the function body.

```
def express_checkout(product_to_quantity: dict[str, int]) -> bool:
    """Return True if and only if the grocery order in product_to_quantity
    qualifies for the express checkout.  product_to_quantity maps products
    to the numbers of those items in the grocery order.

    >>> express_checkout({'banana': 3, 'soy milk': 1, 'peanut butter': 1})
```

```
    >>> express_checkout({'banana': 3, 'soy milk': 1, 'twinkie': 5})
```

```
    """
```

4. As part of a study funded by a major shoe company, we crouched at the finish line of the Boston marathon and kept an ordered list of the shoes that we saw as they passed the finish line. Write a function that, given such a list, will return a dictionary mapping shoe companies to a list of the placements achieved by runners wearing shoes made by those companies.

```
def build_placements(shoes: list[str]) -> dict[str, list[int]]:
    """Return a dictionary where each key is a company and each value is a
    list of placements by people wearing shoes made by that company.

    >>> result = build_placements(['Saucony', 'Asics', 'Asics', 'NB', 'Saucony', \
    'Nike', 'Asics', 'Adidas', 'Saucony', 'Asics'])
    >>> result == {'Saucony': [1, 5, 9], 'Asics': [2, 3, 7, 10], 'NB': [4], \
    'Nike': [6], 'Adidas': [8]}
    True
    """
```