(Last) Surname: _____ (First) Given name: _____

NetID (email): _____ @wisc.edu

Fill in these fields (left to right) on the scantron form (use #2 pencil):
1. LAST NAME (surname) and FIRST NAME (given name), fill in bubbles
2. IDENTIFICATION NUMBER is your Campus ID number, fill in bubbles
3. Under *ABC* of SPECIAL CODES, write your lecture number, fill in bubbles:
   - 001 - MWF 08:50 AM (Mike)
   - 002 - MWF 11:00 AM (Mike)
   - 003 - MWF 09:55 AM (Louis)
   - 004 - MWF 01:20 PM (Louis)
4. Under **F** of SPECIAL CODES, write *C* and fill in bubble **8**

----

**If you miss step 4 above (or do it wrong), the system may not grade you against the correct answer key, and your grade will be no better than if you were to randomly guess on each question. So don't forget!**

----

You may only reference your note sheet. You cannot use books, your neighbors, calculators, or other electronic devices during this exam. Please place your student ID face up on your desk. Turn off and put away portable electronics (including smart watches) now.

Use a #2 pencil to mark all answers. When you're done, please hand in the exam, note sheet, and filled-in scantron form. The note sheet will not be returned.

# Steam Games

The following functions are the same as in P5.

- `project.get_name(idx: int)`: The name of the game at row `idx`

- `project.get_publisher(idx: int)`: The publisher of the game in row `idx`

- `project.get_release_date(idx: int)`: The release date of the game in row `idx` in mm/dd/yyyy format

- `project.get_avg_playtime(idx: int)`: The average playtime (in hours) of the game at row `idx`

- `project.get_price(idx: int)`: The price of the game at row `idx`, like $19.99

- `project.get_positive_reviews(idx: int)`: The number of positive reviews for the game at row `idx`, like 2.81K

- `project.get_negative_reviews(idx: int)`: The number of negative reviews for the game at row `idx`, like 1.13K

As in P5, the dataset has the columns `name`, `publisher`, `release_date`, `avg_playtime`, `price`, `positive_reviews`, and `negative_reviews`.

1. What should the `???` be replaced with to accurately find the the highest average playtime of any game in the dataset?

```
highest_avg_playtime = 0
for idx in range(project.count()):
    if ??? > highest_avg_playtime:
        highest_avg_playtime = project.get_avg_playtime(idx)
```

    A. `project.count()`

    B. `highest_avg_playtime`

    **C.** `project.get_avg_playtime(idx)`

    D. `project.get_avg_playtime(project.count())`

    E. None of the above

Use the following function definitions and dataset to answer the next two questions.

```
def get_year(date):
    return int(date[6:])

def get_year_total(year):                                   # line 1
    games_released = 0                                      # line 2
    for idx in range(project.count()):                      # line 3
        if get_year(project.get_release_date(idx)) == year: # line 4
            games_released += 1                             # line 5
    return games_released                                   # line 6
```

| Name | Release Date | Avg Playtime | Price | Pos. Reviews | Neg. Reviews |
|---|---|---|---|---|---|
| DOOM Eternal | 03/19/2020 | 456 | $39.99 | 165.67K | 16.43K |
| The Sims 4 | 06/18/2020 | 257 | $0 | 106.14K | 15.12K |
| Monster Hunter: World | 08/08/2018 | 3671 | $29.99 | 351.26K | 53.27K |
| Bloons TD 6 | 12/17/2018 | 1551 | $13.99 | 281.79K | 7.5K |
| Team Fortress 2 | 10/10/2007 | 422 | $0 | 954.93K | 61.62K |

2. What does `get_year_total(2020)` return when run on the mini dataset above?

    A. `["DOOM Eternal", "The Sims 4"]`

    B. `1`

    **C.** 2

    D. `713`

    E. None of the above

3. If the `==` is changed to `>=` on line 4 of the above code, what will `get_year_total(2018)` return when run on the mini dataset above?

    A. `["DOOM Eternal", "The Sims 4", "Monster Hunter:  World", "Bloons TD 6"]`

    B. `["Team Fortress 2"]`

    C. `5`

    **D.** 4

    E. None of the above

4. What should the ??? be replaced with to find the game with the most positive reviews that is published within the range of `year1` and `year2` (both inclusive)? Use the included definition of `format_num_reviews`.

```python
def format_num_reviews(num_reviews):
    last_char = num_reviews[-1]
    if last_char == "M":
        review_num = float(num_reviews[:-1]) * 1e6
    elif last_char == "K":
        review_num = float(num_reviews[:-1]) * 1e3
    else:
        review_num = float(num_reviews)
    return round(review_num)

def best_in_range(year1, year2):
    best_idx = None
    best_num_reviews = None
    for idx in range(num_games):
        release_year = get_year(project.get_release_date(idx))
        if year1 <= release_year <= year2:
            num_reviews = format_num_reviews(project.get_positive_reviews(idx))
            if ??? or best_num_reviews < num_reviews:
                best_idx = idx
                best_num_reviews = num_reviews
    return best_idx
```

   A. `best_num_reviews == num_reviews`

   B. `release_year >= year1`

   C. `best_idx != best_num_reviews`

   **D.** `best_idx == None`

   E. None of the above

# Employees

Below is the data that will be used in the following questions.

```
employees = [
    {
        "Name": "John",
        "Age": 45,
        "Salary": 85000,
        "Departments": ["HR", "Finance"]
    },
    {
        "Name": "Mia",
        "Age": 34,
        "Salary": 92000,
        "Departments": ["IT", "Operations"]
    },
    {
        "Name": "Raj",
        "Age": 29,
        "Salary": 64000,
        "Departments": ["Sales"]
    },
    {
        "Name": "Esther",
        "Age": 40,
        "Salary": 105000,
        "Departments": ["Management", "IT"]
    },
    {
        "Name": "Ahmed",
        "Age": 38,
        "Salary": 78000,
        "Departments": ["Operations", "Sales"]
    },
    {
        "Name": "Olivia",
        "Age": 31,
        "Salary": 67000,
        "Departments": ["Sales", "HR"]
    }
]
```

5. Which of the following would create a list of names of employees who have more than one department?

A. `[employee for employee in employees if len(employee["Departments"]) > 1]`

B. `[employee["Name"] if len(employee["Departments"]) > 1 for employee in employees]`

**C.** `[employee["Name"] for employee in employees if len(employee["Departments"]) > 1]`

D. `[for employee in employees if len(employee["Departments"]) == 2 append employee["Name"]]`

6. What will be printed when the following code is run?

```
age_groups = {}
for item in employees:
    age_bin = (item["Age"] // 10) * 10
    if age_bin not in age_groups:
        age_groups[age_bin] = 0
    age_groups[age_bin] += 1
print(len(age_groups), age_groups[30])
```

A. 3 1

B. 4 2

**C. 3 3**

D. 2 1

7. Fill in the ... to complete a function that recursively counts the total number of department entries across all employees:

```
def count_departments(employees):
    if len(employees) == 0:
        return 0
    else:
        return len(employees[0]["Departments"]) + count_departments(...)
```

**A.** `employees[1:]`

B. `employees`

C. `employees["Departments"]`

D. `employees[0]`

8. Consider the recursive function designed to calculate the average age of employees by breaking down the list into smaller chunks. Which option correctly completes the function to calculate the weighted average of two halves of the employee list?

```
def recursive_average_age(employees):
    if len(employees) == 1:
        return employees[0]["Age"]
    else:
        mid = len(employees) // 2
        left_avg = recursive_average_age(employees[:mid])
        right_avg = recursive_average_age(employees[mid:])
        # Complete the following line
        return ...
```

A. (left_avg + right_avg) / 2

**B.** (left_avg * mid + right_avg * (len(employees) - mid)) /
len(employees)

C. (left_avg * len(employees) + right_avg * len(employees)) / 2

D. (left_avg * (len(employees) - mid) + right_avg * mid) /
len(employees)

## General

Use the following code to answer the next two questions.

```
s1 = "abc"
s2 = s1
s1 = s1 + s2 + "xy"
```

9. What is the output of the following code?

   ```
   print(s1)
   ```

   A. `abcabc`

   **B. `abcabcxy`**

   C. `abcxy`

   D. Runtime error

   E. None of the above

10. What is the output of the following code?

    ```
    print(s2)
    ```

    **A. `abc`**

    B. `abcabcxy`

    C. `abcxy`

    D. Runtime error

    E. None of the above

11. What is the output of the following code?

```
data = [["Name", "Age", "State"],
        ["Andrew", 25, "WI"],
        ["Brielle", 21, "MN"],
        ["Cindy", 22, "OH"]]

header = data[0]
data = data[1:]

print(data[-1][header.index("Age")])
```

    A. 21

    **B. 22**

    C. "Age"

    D. Syntax error

    E. None of the above

Use the following code to answer the next two questions.

```
dino = "velociraptor"
d = {}

for c in dino:
    if c not in d:
        d[c] = 1
    else:
        d[c] += 1
```

12. What is the output of the following code?

```
print(d["o"])
```

    A. 1

    **B. 2**

    C. 3

    D. Syntax error

    E. None of the above

13. What is the output of the following code?

```
print(d["c"] + d["r"])
```

    A. 1

    B. 2

    **C.** 3

    D. Syntax error

    E. None of the above

Use the following code to answer the next two questions.

```
fantasy_characters = {
    "Elandor": {
        "species": "Elf",
        "age": 120,
        "skills": ["Archery", "Magic", "Healing"]
    },
    "Thrain": {
        "species": "Dwarf",
        "age": 150,
        "skills": ["Blacksmithing", "Mining", "Brawling"]
    },
    "Lyra": {
        "species": "Human",
        "age": 25,
        "skills": ["Diplomacy", "Swordsmanship", "Stealth"]
    },
    "Zephyr": {
        "species": "Dragon",
        "age": 500,
        "skills": ["Flight", "Fire Breathing", "Intimidation"]
    }
}
```

14. How can I find out how old Zephyr is?

    A. `fantasy_characters["Zephyr"]`

    **B.** `fantasy_characters["Zephyr"]["age"]`

    C. `fantasy_characters["age"]["Zephyr"]`

    D. `fantasy_characters.age_of("Zephyr")`

    E. None of the above

15. How can I find Elandor's species?

     A. `fantasy_characters.get("Elandor").get("species")`

     B. `fantasy_characters["species"]["Elandor"]`

     C. `fantasy_characters.get("species").get("Elandor")`

     D. `fantasy_characters["Elandor"]["species"]`

     **E. A and D**

Use the following code to answer the next two questions.

```
animals = ["Tiger", "Wolf", "Zebra", "Ostrich", "Panda", "Narwhal"]
```

16. What is the output of the following code?

```
new_list = [animal for animal in animals if "a" in animal]
print(new_list)
```

     **A. `["Zebra", "Panda", "Narwhal"]`**

     B. `["Tiger", "Wolf", "Zebra", "Ostrich", "Panda", "Narwhal"]`

     C. `[]`

     D. `["Zebra"]`

     E. None of the above

17. What is the output of the following code?

```
another_new_list = [len(animal) for animal in animals]
print(another_new_list)
```

     A. `["Tiger", "Wolf", "Zebra", "Ostrich", "Panda", "Narwhal"]`

     B. `[7, 7, 7, 7, 7, 7]`

     C. `[5, 5, 5, 5, 5, 5]`

     **D. `[5, 4, 5, 7, 5, 7]`**

     E. `[5, 5, 7]`

18. After executing the following code, what will be the content of `complex_structure`?

```python
import copy
complex_structure = {
    "key1": ["item1", {"subkey": "subitem"}],
    "key2": (42, [28, "deep"])
}
copied_structure = copy.copy(complex_structure)
copied_structure["key1"][1]["subkey"] = "modified"
copied_structure["key3"] = [18,[19,"deep too"]]
```

    **A.** `{"key1": ["item1", {"subkey": "modified"}], "key2": (42, [28, "deep"])}`

    B. `{"key1": ["item1", {"subkey": "modified"}], "key2": (42, [28, "deep"]), "key3": [18, [19, "deep too"]]}`

    C. `{"key1": ["item1", {"subkey": "subitem"}], "key2": (42, [28, "deep"])}`

    D. Syntax error

    E. None of the above

19. Given a sorted array of integers `arr = [2, 4, 4, 6, 6, 6, 7]` and a search function defined as follows:

```python
def search(arr, low, high, target):
    if high < low:
        return -1  # Element is not present in the array
    mid = low + (high - low) // 2
    if target == arr[mid]:
        if mid == len(arr) - 1 or arr[mid + 1] != target:
            return mid
        else:
            return search(arr, mid + 1, high, target)
    elif target < arr[mid]:
        return search(arr, low, mid - 1, target)
    else:
        return search(arr, mid + 1, high, target)
```

What is the result of the function call `search(arr, 0, len(arr) - 1, 6)`?

    A. 0

    **B.** 5

    C. 3

    D. -1

20. Which of the following statements best describes the necessary components for a well-defined recursive function in programming?

    A. A recursive function must include at least two recursive calls within its body to ensure comprehensive exploration of all possible outcomes.

    **B. A recursive function must always have a base case that stops the recursion and a recursive case that continues the recursion.**

    C. A recursive function should only return a single value or result to be considered valid and efficient.

    D. A recursive function requires external variables to track the depth of recursion to prevent stack overflow errors.

21. How can we sort the following list of dictionaries by the number of keys in ascending order? For dictionaries that have the same number of keys, we want to sort them by the total of their values (again, in ascending order).

```
d_list = [{"abc": 3}, {"a": 1, "b": 2, "cde": 0}, {"ab": 4}]
```

    A. `sorted(d_list, key=lambda x:  (len(list(x.keys())), list(x.values())))`

    B. `sorted(d_list, key=lambda x:  (sum(len(k) for k in x), sum(x.values())))`

    C. `sorted(d_list, key=lambda x:  (len(x), x.values()))`

    **D.** `sorted(d_list, key=lambda x:  (len(x), sum(x.values())))`

22. Given the following code:

```python
fruits = "apple,banana,melon"
my_list = fruits.split(",")
my_list[-1] = "kiwi"
new_list = my_list[:2]
my_list.append("grape")

for fruit in my_list:
    if "a" in fruit:
        print(fruit, end=" ")

result = " | ".join(new_list)
print(",", result)
```

What will be the output of this code?

      A. `apple banana grape , banana | kiwi`

      B. `apple banana kiwi grape , apple | banana`

      **C.** `apple banana grape , apple | banana`

      D. `apple banana grape , apple | banana | kiwi | grape`

23. Given the initial contents of the JSON file `"score_history.json"`:

```
{"Bob": [20.0, 10.0],
"Alice": [30.0, 20.0],
"Mike": [100.0, 10.0]}
```

When the following Python code is executed in the same directory as this JSON file, what would be the output that is printed?

```python
def read_json(path):
    with open(path, encoding="utf-8") as f:
        return json.load(f)

def write_json(path, data):
    with open(path, "w", encoding="utf-8") as f:
        json.dump(data, f, indent=2)

scores_dict = read_json("score_history.json")
scores_dict["Louis"] = [50.0, 20.0]
write_json("score_history2.json", scores_dict)
print(scores_dict)
```

    **A.** `{"Bob": [20.0, 10.0], "Alice": [30.0, 20.0], "Mike": [100.0, 10.0], "Louis": [50.0, 20.0]}`

    B. `{"Bob": [20.0, 10.0], "Alice": [30.0, 20.0], "Mike": [100.0, 10.0]}`

    C. `{"Bob": [20.0, 10.0], "Alice": [30.0, 20.0], "Mike": [100.0, 10.0], "Louis": [50.0, 25.0]}`

    D. `{"Bob": [10.0, 20.0], "Alice": [20.0, 30.0], "Mike": [10.0, 100.0], "Louis": [20.0, 50.0]}`

24. Consider the following code and its behavior:

```
x = [1, 2, 3]
y = x
z = x[:]
w = z
x.append(4)
z.append(4)
```

Which of the following statements is correct regarding the relationships and values of `x`, `y`, `z`, and `w`?

      A. `x == y`, `z == w`, but `x is not y` and `z is w`, with `y` and `w` holding different values.

      **B. `x is y`, `z is w`, but `x is not z` and `w is not y`.**

      C. `x == z`, `y is x`, but `z != w`, and `x`, `y`, `z`, and `w` all hold the same values.

      D. `y is x`, `z == x`, and `w` holds a reference to a new list, with values different from `x` and `z`.

25. Suppose you have the following code:

```
Person = namedtuple("Person", ["name", "age"])
john = Person(name="John Doe", age=25)
jane = Person(name="Jane Smith", age=30)

# ??? - Replace with a line of code that causes an AttributeError
print(jane)
```

Which line, if added at line ???, would result in an `AttributeError` due to immutability?

      **A. `jane.age = 31`**

      B. `print(jane.age + 1)`

      C. `john = Person("John Smith", john.age + 1)`

      D. `print(jane.name)`

## Power Generators

The next two questions will use function `find_entities_with_phrase(phrase)`, which returns a list of unique entity names containing a specific phrase, ignoring case-sensitivity:

```python
def find_entities_with_phrase(phrase):
    entity_names = []
    for i in range(len(csv_rows)):
        if phrase.lower() in cell(i, "entity_name").lower():
            entity_names.append(cell(i, "entity_name"))
    return list(set(entity_names))
```

26. Assume you want to identify all entities containing both "River" and "Power" in their names. Which of the following code snippets will fill in **???** to accurately find and return this list?

```python
def find_river_power_entities():
    river_entities = find_entities_with_phrase("River")
    power_entities = find_entities_with_phrase("Power")
    ???
    return sorted(river_power_entities)
```

    **A. river_power_entities = []**
        **for entity in river_entities:**
            **if entity in power_entities:**
                **river_power_entities.append(entity)**

    B. river_power_entities = list(set(river_entities + power_entities))

    C. river_power_entities = river_entities + power_entities

    D. river_power_entities = sorted(river_entities + power_entities)

27. Which of the following statements about the behavior of function `find_entities_with_phrase(phrase)` is **FALSE**?

    A. It returns a list of unique entity names containing the given phrase, ignoring case.

    B. If the given phrase is an empty string `""`, it returns a list of all unique entity names in the dataset.

    **C. The order of the returned list is guaranteed to match the order of entities in the dataset.**

    D. The function uses case-insensitive comparison to match the phrase within entity names.

# Movies

Each entry in the movie database is a dictionary with the following keys and example values:

```
movies = [
    {
        "title": "Inception",
        "year": 2010,
        "duration": 148,
        "genres": ["Action", "Sci-Fi"],
        "rating": 8.8,
        "directors": ["nm0634240"],
        "cast": ["nm0000138", "nm0330687"]
    },
    {
        "title": "The Matrix",
        "year": 1999,
        "duration": 136,
        "genres": ["Action", "Sci-Fi"],
        "rating": 8.7,
        "directors": ["nm0905152", "nm0905154"],
        "cast": ["nm0000206", "nm0005251"]
    },
    {
        "title": "Interstellar",
        "year": 2014,
        "duration": 169,
        "genres": ["Adventure", "Drama", "Sci-Fi"],
        "rating": 8.6,
        "directors": ["nm0634240"],
        "cast": ["nm0000138", "nm0614165"]
    },
    {
        "title": "Parasite",
        "year": 2019,
        "duration": 132,
        "genres": ["Comedy", "Drama", "Thriller"],
        "rating": 8.6,
        "directors": ["nm10014057"],
        "cast": ["nm0437287", "nm3600677"]
    }
]
```

28. Which of the following code snippets correctly creates a list of the titles of Action movies with a rating higher than 7.5?

   **A.** `action_high_rated = [movie["title"] for movie in movies if "Action" in movie["genres"] and movie["rating"] > 7.5]`

   B. `action_high_rated = [movie["title"] for movie in movies if "Action" in genres and movie["rating"] > 7.5]`

   C. `action_high_rated = [movie["title"] for movie in movies if movie["genres"] == "Action" and movie["rating"] > 7.5]`

   D. `action_high_rated = [movie for movie in movies if "Action" in movie["genres"] and movie["rating"] > 7.5]`

29. Which of the following code snippets is equivalent to the function below?

```
def find_sci_fi_movies(movies):
    sci_fi_movies = []
    for movie in movies:
        if "Sci-Fi" in movie["genres"]:
            sci_fi_movies.append(movie["title"])
    return sci_fi_movies
```

   **A.** `[movie["title"] for movie in movies if "Sci-Fi" in movie["genres"]]`

   B. `[movie for movie in movies if "Sci-Fi" in movie["genres"]]`

   C. `{"Sci-Fi":  movie["title"] for movie in movies }`

   D. `for movie in movies:  if "Sci-Fi" in movie["genres"]:`
      `print(movie["title"])`

30. What does this function primarily achieve?

```python
def X(movies):
    result = {}
    for movie in movies:
        for genre in movie["genres"]:
            if genre not in result:
                result[genre] = {"total_rating": 0, "movies_count": 0}
            result[genre]["total_rating"] += movie["rating"]
            result[genre]["movies_count"] += 1
    for genre, data in result.items():
        if data["movies_count"] > 0:
            data["average_rating"] = data["total_rating"] / data["movies_count"]
        else:
            data["average_rating"] = None
    return result
```

    A. It removes all genres from the movies that have below average ratings.

    **B. It calculates the average rating for each genre and records the number of movies in each genre.**

    C. It normalizes the ratings of movies across different genres.

    D. It distributes movies into genres without evaluating their performance.

    E. It bucketizes the movies by genre.

# Blank Page: this page is intentionally blank