CS 220 - Spring 2025

Instructors: Mike Doescher and Louis Oliphant

Final Exam — 12%

(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 **A** and fill in bubble **6**

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!

Many of the problems in this exam are related to the course projects, but some questions assume the availability of slightly different functions (e.g., for accessing the data). We won't have any trick questions where we call a function that doesn't exist, and you need to notice. Thus, if you see a call to a function we haven't explicitly defined in the problem, assume the function was properly implemented (perhaps immediately before the code snippet we DO show) and is available to you.

You may only reference one sheet of notes. You may not use books, your neighbors, calculators, or other electronic devices on 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 this exam and your note sheet in addition to your filled-in scantron.

General Part 1

1. What is the output of the following code?

```
def func1(g, h):
    print((g + h) / 2)

def func2(h, i=5):
    func1(i, h+10)

def func3(g, h):
    func2(g)

g = 5
h = 10
func3(g, h)
```

   A. 7.5

   B. 10.0

   C. 12.5

   D. 15.0

   E. 17.5

Consider the following code for the next 2 questions.

```
def func(a, b, c):
    if a > b:
        if a < c:
            return a
        elif b > c:
            return b
        else:
            return c
    else:
        if a > c:
            return a
        elif b < c:
            return b
        else:
            return c

print(func(10, 3, 5))
```

2. What is the output of the above code?

    A. 5

    B. 3

    C. None

    D. 10

    E. True

3. Which of the following mathematical expressions does function `func` implement?

    A. `min(a, b, c)`

    B. `max(a, b, c)`

    C. `sorted([a, b, c])[1]`

    D. `(a > b) and (a < c)`

    E. `None of the above`

4. Which of the following code snippets will produce this output? Note that the spaces between `"I"` and `"Love"` is equivalent to one tab.

```
I   Love "CS220"...
```

   A. `print("I\tLove \"CS220\"", end="...")`
   B. `print("I\nLove \"CS220\"...")`
   C. `print("I/nLove /"CS220/"")`
   D. `print("I\tLove 'CS220'...")`
   E. `print("I/tLove /"CS220/"", end="...")`

5. What does this expression evaluate to?

```
(not (4 * 2 == 9) and 10 % 3 < 2) or (7 // 3 == 2 and not (5 != 5 or 6 > 9))
```

   A. False
   B. True
   C. None
   D. Error

General Part 2

6. What is the output of the following code snippet?

```
def func(n):
    if n >= 2:
        return func(n // 2) + func(n // 4)
    else:
        return n + 3

print(func(7))
```

   A. 8
   B. 9
   C. 10
   D. 11
   E. 12

7. What is the output of the following code snippet?

```
x = ["a", "b", "c"]
y = ["d", "e", "f"]

z = x + y
x[0] = "w"
print(z[0] + y[-2])
```

   A. we
   B. wd
   C. ae
   D. ad
   E. The code will produce an error

8. Which of the following will return a sorted version of `athlete_list` in descending order of the age of each player?

```
athlete_list = [{"Name": "Sinner", "Age": 23},
{"Name": "Alcaraz", "Age": 21},
{"Name": "Fritz", "Age": 27},
{"Name": "de Minaur", "Age": 26}]
```

   A. `sorted(athlete_list, key = lambda x:  x["Age"])`
   B. `sorted(athlete_list, key = lambda x:  x["Age"], reverse=True)`
   C. `sorted(athlete_list, key = lambda f:  f[1])`
   D. `sorted(athlete_list, key = lambda f:  f[1], reverse=True)`
   E. `sorted(athlete_list.items(), key = lambda x:  x["Age"], reverse=True)`

9. What is the output of the following code snippet?

```
my_list = [["Loki", "Laufeyson"], ["Reed", "Richards"], ["Tony", "Stark"],
["Bruce", "Banner"]]

new_list = [x[-2] for x in my_list if len(x[1]) % 2 == 0]
print(new_list)
```

   A. `["Reed", "Bruce"]`
   B. `["Loki", "Tony"]`
   C. `["Loki", "Reed", "Bruce"]`
   D. `["Reed", "Tony", "Bruce"]`
   E. None of the above

10. What should the `???` be replaced with so that the code below outputs `Student(firstname="Peter", lastname="Parker", year="Graduated!")`?

```
from collections import namedtuple
Student = namedtuple("Student", ["firstname", "lastname", "year"])

Spidey = Student("Peter", "Parker", "Senior")

def update_year(student):
    updates = {"Freshman": "Sophomore", "Sophomore": "Junior",
    "Junior": "Senior", "Senior": "Graduated!"}

    ???

Spidey = update_year(Spidey)
print(Spidey)
```

A. `return Student(student.firstname, student.lastname, updates[student.year])`

B. `student.year = updates[student.year]`
`return student`

C. `student[year] = updates[student[year]]`
`return student`

D. `return Student("Peter", "Parker", updates[student[year]])`

E. `None of the above will work`

General Part 3

The following code will be used to answer the next 3 questions.

```
from bs4 import BeautifulSoup

html_string = """
<html lang="en">
  <head>
    <title>Semantic Test</title>
  </head>
  <body>
    <section>
      <h1>Welcome</h1>
      <article>
        <header>
          <h1>Breaking News</h1>
        </header>
        <p>The latest updates from the tech world...</p>
      </article>
    </section>
    <footer>
      <nav>
        <ul>
          <li><a href="/about">About</a></li>
          <li><a href="/contact">Contact</a></li>
        </ul>
      </nav>
      <div class="contacts">
        <p class="person">Alice Johnson - alice@example.com</p>
        <p class="person">Bob Smith - bob@example.com</p>
      </div>
    </footer>
  </body>
</html>
"""
```

11. What is the value of result after running the following code?

```
bs_obj = BeautifulSoup(html_string, "html.parser")
result = bs_obj.find("h1").get_text()
```

A. Breaking News

B. <h1>Breaking News</h1>

C. Welcome

D. WelcomeBreaking News

E. None of the above

12. Which of the following is the correct code to extract `"Contact"` from the HTML string?

```
bs_obj = BeautifulSoup(html_string, "html.parser")
contact = ...
```

A. `bs_obj.find("a").get_text()`

B. `bs_obj.find_all("a")[1].get_text()`

C. `bs_obj.select("footer")[0].get_text()`

D. `bs_obj.find_all("li")[0].get_text()`

E. `bs_obj.select("a")[0].string`

13. Which of the following codes correctly extracts a list of first names from the contacts section? (Expected output is `["Alice", "Bob"]`)

```
bs_obj = BeautifulSoup(html_string, "html.parser")
result = ...
```

A. `[p.get_text() for p in bs_obj.find_all("p")]`

B. `[p.get_text().split()[0] for p in bs_obj.find_all("p")[-2:]]`

C. `[p.get_text().split(" - ")[0] for p in bs_obj.find_all("p")][-2:]`

D. `[p.get_text().split()[1] for p in bs_obj.find_all("p")]`

E. None of the above

Suppose we have a DataFrame in a variable named `df` and a SQL table containing the same data also named `df`. The following table displays the first 10 rows of the DataFrame. We will use this for the next 3 questions.

student_id	course_grade	midterm1_grade	midterm2_grade	final_exam_grade
1001	87	82	89	90
1002	73	70	68	75
1003	91	88	90	95
1004	65	60	70	66
1005	78	75	80	82
1006	84	85	83	86
1007	92	90	94	93
1008	59	55	60	58
1009	88	85	87	90
1010	76	70	78	79

14. Which of the following SQL queries is **logically equivalent** to the Pandas command:

```
df.groupby("student_id")["course_grade"].mean()
```

A. `SELECT student_id, MEAN(course_grade) FROM df GROUP BY student_id`

B. `SELECT AVG(course_grade) FROM df GROUP BY student_id`

C. `SELECT student_id FROM df WHERE course_grade = AVG(course_grade)`

D. `SELECT student_id, course_grade FROM df WHERE course_grade = AVG(course_grade)`

E. `None of the Above`

15. Which of the following queries will select the `student_id` of students who scored below 50 on midterm 2 from the above `df`?

A. `SELECT student_id FROM df WHERE midterm2_grade > 50`

B. `SELECT student_id FROM df WHERE midterm2_grade < 50`

C. `SELECT student_id FROM df WHERE midterm2_grade <= 50 GROUP BY student_id`

D. `SELECT student_id, midterm2_grade FROM df HAVING midterm2_grade < 50`

E. `WHERE course_grade < 50 SELECT student_id FROM df`

16. Which of the following SQL queries creates a new dataset with all columns from `df` and adds a new column named `passed` that indicates whether the student has course grade greater than 50?

    A. `SELECT *, course_grade > 50 AS passed FROM df`

    B. `SELECT student_id, course_grade > 50 AS passed FROM df`

    C. `SELECT *, course_grade >= 50 FROM df`

    D. `SELECT student_id, passed FROM df WHERE course_grade > 50`

    E. `None of the above`

Suppose we have the below dataframe called `df2` and the matching SQL table that contains personal information about the students. We will use it for the next 2 questions.

student_id	level	major
1001	Junior	Computer Science
1002	Sophomore	Data Science
1003	Senior	Computer Science
1004	Freshman	Psychology
1005	Junior	Data Science
1006	Senior	Statistics
1007	Senior	Computer Science
1008	Freshman	Economics
1009	Junior	Computer Science

17. Which of the following SQL queries correctly **counts the number of students in each major** and **returns only the majors that have more than 2 students**?

    A. `SELECT major, COUNT(*) AS num_students FROM df2 HAVING num_students > 2 GROUP BY major;`

    B. `SELECT major, COUNT(student_id) AS num_students FROM df2 WHERE COUNT(student_id) > 2 GROUP BY major;`

    C. `SELECT major, COUNT(*) AS num_students FROM df2 GROUP BY major HAVING num_students > 2;`

    D. `SELECT major, COUNT(student_id) AS num_students FROM df2 GROUP BY major WHERE COUNT(*) > 2;`

    E. `None of the Above`

18. Which of the following SQL queries will return the level with the most students?

    A. `SELECT level FROM df2 GROUP BY level ORDER BY COUNT(student_id) DESC LIMIT 1;`

    B. `SELECT level, COUNT(student_id) FROM df2 GROUP BY level HAVING COUNT(student_id) = MAX(COUNT(student_id));`

    C. `SELECT level, COUNT(student_id) FROM df2 ORDER BY COUNT(student_id) DESC LIMIT 1;`

    D. `SELECT level FROM df2 GROUP BY level HAVING COUNT(student_id) > 5;`

    E. `SELECT level FROM df2 WHERE COUNT(student_id) = MAX(COUNT(student_id));`

Suppose we have the following dataset stored in a Pandas DataFrame called `sales_df`, showing monthly sales for two different products with month as the index.

month	**product_a_sales**	**product_b_sales**
Jan	120	150
Feb	135	160
Mar	150	170
Apr	140	165
May	160	175
Jun	170	180

19. You are asked to create a **line plot** that shows the **sales trend for both products** over the months. Which of the following first lines of code will correctly create the desired plot?

    The following plotting code is given:

    ```
    import matplotlib.pyplot as plt

    # [YOUR FIRST LINE OF CODE HERE]
    ```

    A. `sales_df.plot.line()`

    B. `plt.plot(sales_df["product_a_sales"], sales_df["product_b_sales"])`

    C. `sales_df.plot.line(x="product_a_sales", y="product_b_sales")`

    D. `plt.line(sales_df["month"], sales_df["product_a_sales"], sales_df["month"], sales_df["product_b_sales"])`

    E. `None of the Above`

Suppose we have the following dataset stored in a Pandas DataFrame called `revenue_df`, showing quarterly revenue for three departments.

department	Q1	Q2	Q3	Q4
Electronics	500	550	600	650
Clothing	300	320	330	360
Grocery	400	420	450	470

20. You are asked to create a **bar plot** that shows the **Q1 revenue for each department**. Which of the following first lines of code will correctly create the desired plot?

    The following plotting code is given:

    ```
    import matplotlib.pyplot as plt

    # [YOUR FIRST LINE OF CODE HERE]
    ```

    A. `revenue_df.plot.bar()`

    B. `plt.plot.bar(revenue_df["department"], revenue_df["Q1"])`

    C. `revenue_df.plot.bar(x="department", y=["Q1"])`

    D. `plt.bar(revenue_df["Q1"], revenue_df["department"])`

    E. `None of the Above`

21. What is the difference between a Pandas `DataFrame` and a Pandas `Series`?
    Choose all the correct options:

    I. `A Pandas DataFrame can only have a single column, while a Pandas Series can have multiple columns.`

    II. `A Pandas DataFrame can only contain a single data type, while a Pandas Series can contain multiple data types.`

    III. `A Pandas Series represents a single cell within a DataFrame.`

    IV. `A Pandas DataFrame consists of one or more columns, while a Pandas Series can only have a single column.`

    A. `I, II, and III`

    B. `I`

    C. `II and IV`

    D. `III and IV`

    E. `IV`

22. Consider the following `DataFrame`.

```
|    | name    | age | city         |
|----|---------|-----|--------------|
| 1  | Alice   | 24  | New York     |
| 2  | Mike    | 67  | Los Angeles  |
| 3  | Louis   | 76  | Chicago      |
```

Which command correctly selects Alice's age?

A. `df.loc[0]["age"]`

B. `df.iloc[1]["age"]`

C. `df.loc[1]["age"]`

D. `df.loc["Alice"]["age"]`

E. `df.iloc[0][2]`

For the next question, please consider the following code:

```
import pandas as pd
data = {"Title": ["Inception", "Interstellar", "The Prestige", "Oppenheimer"],
"Release Year": [2010, 2014, 2000, 2023],
"Total Gross (M)": [839, 731, 110, 958]}

movies_df = pd.DataFrame(data)
```

23. Which option below will correctly produce a DataFrame that only contains the movies which grossed over $750M and were released before 2020?

A.
```
movies_df[(movies_df["Total Gross (M)"] > 750) and
(movies_df["Release Year"] < 2020)]
```

B.
```
movies_df[(movies_df["Total Gross (M)"] > 750) or
(movies_df["Release Year"] < 2020)]
```

C.
```
movies_df[(movies_df["Total Gross (M)"] > 750) |
(movies_df["Release Year"] < 2020)]
```

D.
```
movies_df[(movies_df["Total Gross (M)"] > 750) &
(movies_df["Release Year"] < 2020)]
```

E. More than one option is correct

The next question refers to the `piazza_df` DataFrame shown below.

- `name` is a student's nickname/ID.
- `email` is the contact address.
- `role` indicates whether the user is a **student** or **instructor**.
- All remaining columns contain Piazza activity statistics.
- Some values in `name` or `email` are missing (`np.NaN`).

id	name	email	role	answers	followups	replies_to_followups
bfb1	timid city	tc@wisc.edu	student	0	0	0
11f1	hard coffee	hc@wisc.edu	student	0	0	0
1e59	hot love	hl@wisc.edu	student	0	0	0
00fa	funny house	fh@wisc.edu	student	0	0	0
bf76	calm student	cs@wisc.edu	student	0	0	0
c79a	NaN	mn@wisc.edu	student	0	0	0
0b45	cold chair	NaN	student	5	3	1
3h56	NaN	NaN	instructor	2	4	5

24. The following code adds a new `"no_engagement"` column that marks users with no engagement and groups them by `"role"`:

```
piazza_df["no_engagement"] = (piazza_df["answers"] == 0) & \
                             (piazza_df["followups"] == 0) & \
                             (piazza_df["replies_to_followups"] == 0)

result = piazza_df.groupby(["role", "no_engagement"])["name"].count()
```

Which SQL query replicates the logic of the `result` object?

A.
```
SELECT role, COUNT(name)
FROM piazza
WHERE answers = 0 AND followups = 0 AND replies_to_followups = 0
GROUP BY role
```

B.
```
SELECT role,
(answers = 0 AND followups = 0 AND replies_to_followups = 0) AS
no_engagement,
COUNT(name)
FROM piazza
GROUP BY role, no_engagement
```

C.
```
SELECT role, COUNT(name)
FROM piazza
GROUP BY role
HAVING SUM(answers + followups + replies_to_followups) = 0
```

D.
```
SELECT DISTINCT role,
(answers = 0 AND followups = 0 AND replies_to_followups = 0) AS
no_engagement
FROM piazza
ORDER BY role
```

The following two questions are based on the `fruit_data` DataFrame shown below, where `Sales` indicates the quantity sold (in thousands of units), `Price` indicates the price per unit (in hundreds of USD), and `Type` indicates the type of fruit.

Sales	Price	Type
120	1.2	Apple
85	0.9	Orange
150	0.8	Banana
60	2.1	Apple
90	1.5	Orange
110	1.3	Banana

25. What correctly fills in each numbered box to create **separate scatter plots** for each fruit type with:

- Apples as red circles ("o")
- Oranges as green right-pointing triangles (`">"`)
- Bananas as blue squares ("s")

```
types = list(set(fruit_data["Type"]))
colors = ["r", "g", "b"]
markers = ["o", ">", "s"]

for i in range(len( 1 )):
   fruit_type =  2 [i]
   subset = fruit_data[fruit_data["Type"] ==  3 ]
   subset.plot.scatter(
     x="Price", y="Sales",
     color=  4 [i],
     marker=  5 [i]
   )
```

A. [1]: `types`,[2]: `fruit_data["Type"]`, [3]: `"Type"`, [4]: `colors`, [5]: `markers`

B. [1]: `fruit_data["Type"]`, [2]: `types`, [3]: `fruit_type`, [4]: `colors`, [5]: `shapes`

C. [1]: `types`, [2]: `types`, [3]: `fruit_type`, [4]: `col`, [5]: `mark`

D. [1]: `types`, [2]: `types`, [3]: `fruit_type`, [4]: `colors`, [5]: `markers`

E. [1]: `types`, [2]: `types`, [3]: `"Type"`, [4]: `c`, [5]: `m`

26. Define `fruit_sales`, which is the **total sales per fruit type**, as:

```
fruit_sales = fruit_data.groupby("Type")["Sales"].sum()
```

You want to plot **sales in 1000 units** as a **horizontal bar plot** with:

- Figure size: 5 inches wide × 3 inches high
- Bars colored green (`"g"`)
- X-axis label: `"Sales (1000 units)"`
- Title: `"Normalized Fruit Sales"`
- Grid lines shown

Which code snippet achieves this?

A.
```
ax = fruit_sales.plot.barh(figsize=(5, 3))
ax.set_xlabel("Sales (1000 units)")
ax.set_title("Normalized Fruit Sales")
ax.grid()
```

B.
```
ax = (fruit_sales / 1000).plot.barh(figsize=(5, 3), color="g")
ax.set_xlabel("Sales (1000 units)")
ax.grid()
```

C.
```
ax = fruit_sales.plot.barh(figsize=(5, 3), color="g")
ax.set_xlabel("Sales (1000 units)")
ax.set_title("Normalized Fruit Sales")
ax.grid()
```

D.
```
ax = fruit_sales.plot.barh(figsize=(3, 5), color="g")
ax.set_xlabel("Sales (1000 units)")
ax.set_title("Normalized Fruit Sales")
ax.grid()
```

E.
```
ax = (fruit_sales / 1000).plot.barh(figsize=(5, 3), color="g")
ax.set_xlabel("Sales (1000 units)")
ax.set_title("Normalized Fruit Sales")
ax.grid()
```

Movies

Consider the following list of dictionaries for the next 3 questions:

```
movies = [
 {"title": "Zodiac",
  "year": 2007,
  "duration": 157,
  "genres": ["Crime", "Drama", "Mystery"],
  "rating": 7.7,
  "directors": ["David Fincher"],
  "cast": ["Jake Gyllenhaal",
   "Robert Downey Jr.",
   "Mark Ruffalo",
   "Anthony Edwards"]},
 {"title": "Avengers: Infinity War",
  "year": 2018,
  "duration": 149,
  "genres": ["Action", "Adventure", "Sci-Fi"],
  "rating": 8.4,
  "directors": ["Anthony Russo", "Joe Russo"],
  "cast": ["Robert Downey Jr.",
   "Chris Hemsworth",
   "Mark Ruffalo",
   "Chris Evans"]},
 {"title": "Space Jam: A New Legacy",
  "year": 2021,
  "duration": 115,
  "genres": ["Adventure", "Animation", "Comedy"],
  "rating": 4.5,
  "directors": ["Malcolm D. Lee"],
  "cast": ["LeBron James", "Don Cheadle", "Cedric Joe", "Khris Davis"]}
  ]
```

27. Which of the following lines of code will sort `movies` by `duration` in ascending order?

    A. `sorted(movies, key=lambda m:  m["duration"], reverse=True)`

    B. `sorted(movies, key=lambda m:  m["duration"], ascending=True)`

    C. `movies.sort(key="duration")`

    D. `sorted(movies, key=lambda m:  m["duration"])`

    E. `movies.sorted(key=lambda m:  m["duration"])`

28. Which of the following lines of code creates a dictionary that maps movie **titles** to their **ratings** for movies with a **rating** higher than **5**?

    A. `{movie["rating"]: movie["title"] for movie in movies if movie["rating"] > 5}`

    B. `{movie["title"]: movie["rating"] for movie in movies}`

    C. `{movie["title"]: movie["rating"] for movie in movies if movie["rating"] > 5}`

    D. `{movie.title: movie.rating for movie in movies if movie["rating"] > 5}`

    E. `{movie["title"]: movie["rating"] for movie in movies.items() if movie["rating"] > 5}`

29. Consider the following code:

```
final_dict = {}
for movie in movies:
    cat_val = movie[category]
    if cat_val not in final_dict:
        final_dict[cat_val] = []
    final_dict[cat_val].append(movie["title"])
```

What is **TRUE** about the above code? Be careful.

    A. `If category="cast", then the code will raise an error`

    B. `The code will raise an error if movies is empty`

    C. `The dictionary maps categories to a list of movie titles`

    D. `If category="year", then the code will raise an error`

    E. `None of the above`

Stars and Planets

The next three questions combine topics from P10 and P11, covering our study of stars and planets beyond our Solar System.

30. List all CSV files in the `data` directory in reverse-alphabetically sorted order. Choose the best answer.

```
files = os.listdir("data")
csv_files = ???
```

Which replacement for `???` matches the above requirements?

A. `[f for f in files if f.endswith("csv")][:-1]`

B. `sorted([os.path.join("data", f) for f in files if f[-3:]=="csv"], reverse=True)`

C. `[f for f in sorted(files, reverse=True) if "csv" in f]`

D. `sorted([os.path.join("data", f) for f in files if f.endswith("csv")], key=lambda x:  x.split("/")[0], reverse=True)`

E. `sorted([f for f in files if f.endswith("csv")], reverse=True)`

31. Assume that you've already loaded `planets_1.csv` and created a `Planet` namedtuple:

```
planets_attributes = ["planet_name",
                      "host_name",
                      "discovery_method",
                      "discovery_year",
                      "controversial_flag",
                      "orbital_period",
                      "planet_radius",
                      "planet_mass",
                      "semi_major_radius",
                      "eccentricity",
                      "equilibrium_temperature",
                      "insolation_flux"]

Planet = namedtuple("Planet", planets_attributes)
```

Now, you want to **create a scatter plot of planet mass vs. radius** for only planets discovered via the `"Transit"` method, where both values are present and `planet_mass < 250`. What should replace the `???` in the `if` condition in the function below to complete this task?

```
def plot_transit_mass_radius(planets_rows, planets_header):
    x = []
    y = []
    for i in range(len(planets_rows)):
        method = planet_cell(i, "Discovery Method", planets_rows, planets_header)
        mass = planet_cell(i, "Planet Mass [Earth Mass]",
               planets_rows, planets_header)
        radius = planet_cell(i, "Planet Radius [Earth Radius]",
                 planets_rows, planets_header)
        if ???:
            continue
        x.append(mass)
        y.append(radius)
    plot_scatter(x, y, x_label="Planet Mass", y_label="Planet Radius")
```

A. `method != "Transit" or mass is None or radius is None or mass >= 250`

B. `method == "Transit" and mass < 250 and radius != None`

C. `mass is None or radius is None or method is None`

D. `planet_cell(i, "Planet Radius", planets_rows, planets_header) < 250`

E. `method != "Transit" and radius is None and mass is None and mass < 250`

32. Given function `get_all_paths_in()`, which takes a directory's relative path as input and returns a list of relative paths for all files within that directory and its subdirectories.

```
def get_all_paths_in(directory):
    relative_paths = []
    directory_contents = [os.path.join(directory, content) for content in
        os.listdir(directory) if not content.startswith(".")]
    for item_path in directory_contents:
        if os.path.isfile(item_path) == True:
            relative_paths.append(item_path)
        else:
            relative_paths.extend(get_all_paths_in(item_path))
    sorted_relative_paths = sorted(relative_paths,
        key=lambda path: path.split(os.path.sep), reverse=True)
    return sorted_relative_paths
```

The function below uses recursion to count JSON files in nested directories (like `broken_data`), ignoring hidden files (starting with "."). Fill in the missing lines:

```
def count_json_files(directory):
    count = 0
    files = os.listdir(directory)
    for item in sorted([x for x in files if not x.startswith(".")]):
        path = os.path.join(directory, item)
        #### Fill in the missing lines ####
    return count
```

Which option correctly completes the function?

A.
```
if os.path.isfile(path) and item.endswith(".json"):
    count += 1
elif os.path.isdir(path):
    count += count_json_files(path)
```

B.
```
if item.endswith(".json"):
    count += 1
count += count_json_files(path)
```

C.
```
if os.path.isdir(path):
    count += len(get_all_paths_in(path))
elif path.endswith(".json"):
    count += 1
```

D.
```
if os.path.isfile(path):
    count += 1
else:
    count += count_json_files(path)
```

E.
```
count += sum(1 for p in get_all_paths_in(directory) if
p.endswith(".json"))
```

Rankings

In P12 and P13, we analysed World University Rankings data. For the following three questions, assume that you have a pandas DataFrame named `rankings` with the following data:

Year	World Rank	Institution	Country	National Rank	Education Rank	Faculty Rank	Research Rank
2023	28	University of Wisconsin–Madison	USA	20	36	30	41
2023	7	University of Chicago	USA	5	8	22	28
2022	27	McGill University	Canada	2	25	40	44
2022	5	Oxford University	United Kingdom	2	7	9	4
2021	56	Ohio State University	USA	34	124	137	37
2021	13	University of Tokyo	Japan	1	39	107	27

33. What is the output of the following code?

```
rankings[(rankings["Institution"].str.contains("University of")) |
(rankings["World Rank"] < 20)]["Faculty Rank"].min()
```

A. 30

B. 22

C. 40

D. 9

E. 107

34. Which of the following will produce a DataFrame that contains universities whose `Research Rank` rank is worse than their `Education Rank`, sorted in reverse alphabetical order by `Institution`. Note that `1` is the best rank?

A.
```
pd.read_sql("SELECT * FROM rankings WHERE ('Research Rank'-
'Education Rank') > 0 ORDER BY 'World Rank' DESC", conn)
```

B.
```
pd.read_sql("SELECT * FROM rankings WHERE ('Research Rank'-
'Education Rank') < 0 ORDER BY 'Institution' DESC", conn)
```

C.
```
pd.read_sql("SELECT * FROM rankings WHERE ('Education Rank'-
'Research Rank') < 0 ORDER BY 'Institution' ", conn)
```

D.
```
pd.read_sql("SELECT * FROM rankings WHERE 'Education Rank'<
'Research Rank' ORDER BY 'Institution' ", conn)
```

E.
```
pd.read_sql("SELECT * FROM rankings WHERE 'Education Rank'<
'Research Rank' ORDER BY 'Institution' DESC", conn)
```

35. What is the output of the following code?

```
x = pd.read_sql("SELECT * FROM rankings WHERE 'Year' % 2 == 1 AND
'Faculty Rank' <= 50 ORDER BY 'World Rank' ", conn)
print(x["Institution"].iloc[0])
```

A. Oxford University

B. University of Chicago

C. University of Tokyo

D. University of Wisconsin-Madison

E. McGill University

(Blank Page)