

CIS1100.py – Spring 2025 – Exam II

Name: _____ PennID (e.g. 12345): _____

My signature below certifies that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this examination.

Signature

Date

- Do not open this exam until told by the proctor.
- You will have exactly 60 minutes to take this exam.
- Make sure your phone is turned OFF (not on vibrate!) before the exam starts.
- Food and gum are not permitted—don't be noisy or messy.
- You may not use your phone or open your bag for any reason, including to retrieve or put away pens or pencils, until you have left the exam room.
- This exam is closed-book, closed-notes, and closed computational devices.
- If you get stuck on a problem, it may be to your benefit to move on to another question and come back later.
- All code must be written in proper Python format.
- Do not separate the exam pages. Do not take any exam pages with you. The entire exam packet must be turned in as is.
- There are blank pages at the end of the exam if you need extra space for any graded answers.
- Use a pencil, or blue or black pen to complete the exam.
- If you have any questions, raise your hand and a proctor will come to you.
- When you turn in your exam, you may be required to show your PennCard. If you forgot to bring your ID, talk to an exam proctor immediately.
- We wish you the best of luck!

Q1	Q2	Q3	Q4	Q5	Q6
Matching Snippets	OOP Details	Executing Code	Structured Data	Dictionaries	Bonus

Q1. Understanding Code Snippets

Below, we have five snippets listed in the first section. Each of these snippet operates on some list `nums`. For each **original snippet**, your job is threefold:

- Select the single **alternative snippet** that produces exactly the same value of `result`.
- Select which of the corresponding **outputs** would correspond to running this original snippet and printing `result` when `nums = [5, 4, 3, 6, 7]`.
- Compute what the value of `result` would be if `nums = [4, 0, 1, 2]`

Original Snippets:

1.

```
result = []
for num in nums:
    if num % 2 == 0:
        result.append(num)
```

2.

```
result = []
for num in nums:
    result.append(num * 2)
```

3.

```
result = []
for num in nums:
    if num % 2 == 0:
        result.append(num * 2)
```

4.

```
result = nums[0]
for num in nums:
    if num > result:
        result = num
```

5.

```
result = 1
for num in nums:
    if num % 2 == 0:
        result *= num
```

Alternative Snippets:

A.

```
result = list(map(lambda num: num * 2, nums))
```

B.

```
result = [num * 2 for num in nums if num % 2 == 0]
```

C.

```
tmp = filter(lambda x: x % 2 == 0, nums)
result = reduce(lambda x, y: x * y, tmp)
```

D.

```
result = [num for num in nums if num % 2 == 0]
```

E.

```
result = max(nums)
```

Outputs for nums = [5, 4, 3, 6, 7]:

I.	II.	III.	IV.	V.
[10, 8, 6, 12, 14]	[4, 6]	24	7	[8, 12]

Answer Here:

Original Snippet	Matching Alternate (Select A-E)	Result for nums = [5, 4, 3, 6, 7] (Select I-V)	Result for nums = [4, 0, 1, 2] (Write the Value)
1			
2			
3			
4			
5			

Q2. Ticket to Write

This question is about your understanding of how classes are written and used. Here's an example of a class that describes how a `TrainTicket` object behaves. *You don't need to know how the methods here work, so don't spend too much time reading the method bodies here.*

```
class TrainTicket:
    def __init__(self, origin: str, destination: str, date: str,
                 passenger: str):
        self.origin = origin
        self.destination = destination
        self.date = date
        self.passenger = passenger

    def validate(self):
        valid_trip = self.origin != self.destination
        month = int(self.date[0:2])
        day = int(self.date[3:5])
        maybe_valid_date = (1 <= month <= 12) and (1 <= day <= 31)
        return maybe_valid_date and valid_trip

    def transfer(self, recipient):
        self.passenger = recipient

    def issue_return_trip(self, date):
        return TrainTicket(
            self.destination, self.destination, date, self.passenger
        )
```

Question 2.1

What are the names of all of the **attributes** of the `TrainTicket` class? (Sometimes we have referred to these as *fields* or *instance variables*, too.)

Answer:

Question 2.2

What are the names of all of the **methods** of the `TrainTicket` class? (Magic methods are methods, too.)

Answer:

Question 2.3

How would I create a new `TrainTicket` object representing a ticket for "Jude" traveling from "NYC" to "BOS" on "04/07" and save it in a variable `tt`? **Write one statement (i.e. line of code)**, but you can write across multiple lines if you need the space.

Answer:

Question 2.4

Jude is too unwell to travel, and he wants to transfer his ticket to his partner "Willem". How would I modify the `TrainTicket` stored in variable `tt` **using a method** so that the ticket has "Willem" as the passenger? **Write one statement**, but you can write across multiple lines if you need the space.

Answer:

Question 2.5

Willem needs to get home, too. How would I call a method on `tt` create a new `TrainTicket` object for the same person on the date "04/30" that switches the origin and destination and then stores that object in a new variable called `rt`? **Write one statement**, but you can write across multiple lines if you need the space.

Answer:

Q3. What is Printed?

Write the value that gets printed, or write **“error”** if there is an error during the execution of these lines of the program.

Question 3.1

```
names = ["Chris", "Claude", "Toni", "Eloise", "Merril"]
grades = [90, 91, 65, 14, 54]
gradebook = {}
for i, name in enumerate(names):
    grade = grades[i]
    gradebook[name] = grade
print(gradebook[54])
```

Answer:

Question 3.2

```
set_one = set("cis1100isthebest") # adds each character to a set
set_two = set("cis4480isthebest")
result = {char for char in set_one if char not in set_two}
print(result)
```

Answer:

Question 3.3

```
def mystery_foo(input):
    if input == 1:
        return 0

    if input % 2 == 0:
        return 1 + mystery_foo(input // 2)
    else:
        return 1 + mystery_foo(input * 3 + 1)

print(mystery_foo(5))
```

Answer:

Q4. The CIS1100 Navbar

The NavBar at the top of our course website can be modeled cleanly in XML. Here is a file navbar.xml that demonstrates how it is structured.

```
<navbar>
  <tab label="Homework" dropdown="true">
    <menu>
      <item>Homework Table</item>
      <item>HW0: Hello, World!</item>
      <item>HW1: Rivalry</item>
      <item>HW2: Personality Quiz</item>
      <item>HW3: Hail, Caesar!</item>
      <item>HW4: Restaurant Recommendations</item>
      <item>HW5: Furious Flying Fish</item>
      <item>HW6: Emoji Blender</item>
      <item>HW7: Recursion</item>
    </menu>
  </tab>
  <tab label="Schedule" dropdown="false"> </tab>
  <tab label="Staff" dropdown="false"> </tab>
  <tab label="Recitations" dropdown="false"> </tab>
  <tab label="Office Hours" dropdown="false"> </tab>
  <tab label="SRS" dropdown="false"> </tab>
  <tab label="Policies" dropdown="true">
    <menu>
      <item>Grading</item>
      <item>Homework</item>
      <item>Collaboration, AI, and Outside Resources</item>
    </menu>
  </tab>
  <tab label="Exams" dropdown="true">
    <menu>
      <item>Exam 1</item>
      <item>Exam 2</item>
    </menu>
  </tab>
  <tab label="Resources" dropdown="true">
    <menu>
      <item>Getting Help</item>
      <item>Style Guide</item>
    </menu>
  </tab>
  <tab label="Wellness" dropdown="false"></tab>
</navbar>
```

Question 4.1

How many `<tab>` elements are present in the example above?

Answer:

Question 4.2

Each `<tab>` element has two attributes. What are the names of those two attributes?

Answer:

Question 4.3

Some `<tab>` elements contain `<menu>` elements as children. What is the name of the attribute of the `<tab>` element that indicates whether or not that element has any children?

Answer:

Question 4.4

Harry tried to write code that would find the third `<tab>` element (the one labeled “Staff”) inside the soup object `navbar`, but he made an important mistake.

```
1 xmlfile = open("navbar.xml", "r")
2 navbar = BeautifulSoup(xmlfile, "xml")
3 third_tab = navbar.find("tab")[2]
4 print(third_tab)
```

When running this code, we get a `KeyError` on line 3! Rewrite all or part of **line 3** so that this snippet correctly finds and prints that third `<tab>` element.

Answer:

Question 4.5

Harry writes a poorly named function below. This function returns a dictionary, but there are no comments to explain what it does!

```
def process(navbar) -> dict:
    d = {}
    for tab in navbar.find_all("tab"):
        if tab["dropdown"] == "true":
            tab_label = tab["label"]
            d[tab_label] = set()
            for item in tab.find_all("item"):
                d[tab_label].add(item.string)
    return d
```

Assuming that the input navbar is always a BeautifulSoup object (see previous sub-question for an example), the keys of the output dictionary will be strings representing <tab> labels.

What is the **type** of every *value* in this dictionary?

Answer:

We call the function on the XML example provided above like so:

```
xmlfile = open("navbar.xml", "r")
navbar = BeautifulSoup(xmlfile, "xml")
d = process(navbar)
```

Briefly explain why “Exams” would be present as a key in the output dictionary while “SRS” would be absent. A few words should suffice.

Answer:

Write out one full entry (key-value pair) that would be found in the output dictionary. Any format is OK, but make sure to specify which part is the key and which is the value.

Answer:

Q5. Interning for the Registrar

You are tasked with some analysis tasks revolving around **file reading** and **JSON parsing** with respect to Penn's course data! Here's an example JSON file you might parse that represents **two courses**, one with two sections and one with only a single section.

```
1  {
2    "courses": [
3      {
4        "department": "CIS",
5        "course_number": 1200,
6        "course_title": "Programming Languages and Techniques",
7        "sections": [
8          {
9            "section_id": 001,
10           "days": "MWF",
11           "instructors": ["Swapneel Sheth", "Benjamin Pierce"]
12         },
13         {
14           "section_id": 002,
15           "days": "MWF",
16           "instructors": ["Swapneel Sheth", "Benjamin Pierce"]
17         }
18       ]
19     },
20     {
21       "department": "RELS",
22       "course_number": 2560,
23       "course_title": "Existential Despair",
24       "sections": [
25         {
26           "section_id": 001,
27           "days": "M",
28           "instructors": ["Justin McDaniel"]
29         }
30       ]
31     }
32   ]
33 }
```

If we load a JSON file with the structure above into a dict, the "courses" key maps to a list of all the courses. For the questions that follow, make sure to write functions and code that can handle any size JSON file with any number of courses! You may assume that all JSON files are well-formed and that every course contains all of the keys shown above.

Question 5.1 Loading the JSON

First, complete the following function that opens a file specified as a command-line argument, parses its contents using Python's `json` library, and returns the resulting dictionary. Your Python code will be run using the following command:

```
python3 registrar.py <file.json>
```

Here, `file.json` is the name of the file that will be provided for you to read and process.

```
1 import sys
2 import json
3 def load_json(filename):
4     # create a file object to be parsed
5     json_file = ___BLANK_1___(filename, "r")
6     # parse the file's contents using the library
7     return ___BLANK_2___.___BLANK_3___(json_file)
8
9 if __name__ == "__main__":
10    # read the correct command-line arg.
11    filename = sys.__BLANK_4__[___BLANK_5___]
12    all_courses = load_json(filename)
13    ... # the rest omitted for space
```

Blank	Code
___BLANK_1___	
___BLANK_2___	
___BLANK_3___	
___BLANK_4___	
___BLANK_5___	

Question 5.2 Filtering by Departments

Write a function that returns a list of courses, but only if each course belongs to a department included in the set **depts**.

For clarity, here is an example of how the function will be called:

```
depts = {"BE", "CBE", "CIS", "ESE", "MSE", "MEAM"}
json_dict = load_json(fname)
req_courses = filter_department(json_dict, depts)
```

```
def filter_department(json_dict: dict, depts: set[str]) -> list:
    return list(filter(__lambda__, json_dict["courses"]))
```

Write the *Lambda expression* that should be used in the function above in the box below.

Answer:

Question 5.3 How many sections?

You've always wanted to know how many sections are offered across all courses at Penn. Instead of writing an iterative solution, you're going to write one using recursion.

For clarity, here is an example of how the function will be called:

```
json_dict = load_json(fname)
num_sections = count_total_sections(json_dict["courses"]):
```

```
def count_total_sections(courses: list) -> int:
    if __BASE_CASE__:
        return 0
    num_sections = len(__BLANK_A__)
    return num_sections + count_total_sections(__BLANK_B__)
```

Fill in the table below to answer this question.

Blank	Code
__BASE_CASE__	
__BLANK_A__	
__BLANK_B__	

Q6. Bonus

Recommend something for the course staff: a book, a movie, a song, whatever. Or, if not, then you could create a piece of art! All exams will get full credit for this question even if you leave it blank.

Answer:

Extra Work Space

You may use this page for additional space for answers; keep it attached to this exam. Clearly note on the original question page that your answer is on this extra page, and clearly note on this page what question you are answering.

Answer: