Editorial-Solution-W2A1: Federated Learning and Python Game Simulation - Exploring AI Techniques and Data Structure Operations

---

Question 1

What is the primary purpose of Federated Learning ?

A) To share private chats on WhatsApp

B) To make computers learn from data without directly accessing it

C) To create targeted advertisements

D) To build battle games in Python

Correct Answer: B

Explanation:

Federated Learning is described as a way for computers to learn from data without actually seeing the data directly. It's mentioned as an explanation for how devices can seem to know user preferences without directly accessing private conversations

---

Question 2

In the battle game, how is the damage for each attack determined?

A) It's always a fixed value of 15

B) It's randomly generated between 1 and 100

C) It's randomly generated between 10 and 20

D) The player inputs the damage value

Correct Answer: C

Explanation:

The game uses the random.randint(10, 20) function to generate a random integer between 10 and 20 for each attack's damage

---

Question 3

What is the purpose of the time.sleep(2) function in the game code?

A) To pause the game for player input

B) To simulate the generation of an attack and add realism

C) To allow time for the damage calculation

D) To synchronize turns between players

Correct Answer: B

Explanation:

The time.sleep(2) function is used to add a 2-second delay after printing "Generating attack.....", which is intended to make the game feel more realistic by simulating the time taken to generate an attack

---

Question 4:

How does the 'defend' action modify the damage taken in the game?

A) It completely negates the damage

B) It reduces the damage by 25%

C) It reduces the damage by 50%

D) It doubles the damage for the next turn

Correct Answer: C

Explanation:

When a player is defending, the damage they receive is halved. This is implemented in the code as "damage = damage // 2", where // represents integer division, effectively reducing the damage by 50%

Question 5

What Python library is used to create the bar graph for visualizing game simulation results?

A) NumPy

B) Pandas

C) Seaborn

D) Matplotlib

Correct Answer: D

Explanation:

The code uses the matplotlib library to create the bar graph. This is evident from the import statement "import matplotlib.pyplot as plt" and the subsequent use of plt functions to create and customize the graph

---

Question 6

What is the output of

print([i for i in {1:'a', 2:'b'}])

A) [1, 2]

B) ['a', 'b']

C) [(1, 'a'), (2, 'b')]

D) Error

Correct Answer: A

Explanation:

- When iterating through a dictionary, Python by default returns the **keys**
- The comprehension **[i for i in dict]** is equivalent to **list(dict.keys())**
- This dictionary:Contains keys **1** and **2**, resulting in output **[1, 2]**

python{1: 'a', 2: 'b'}

- Key points about dictionary iteration:
    i. Direct iteration: **for key in dict:** → keys
    ii. Explicit methods:
        - **.keys()** → view object of keys
        - **.values()** → view object of values
        - **.items()** → view object of (key, value) pairs
- This behavior is consistent across all Python versions that support dictionary comprehensions

Iterating through a dictionary returns its keys. The comprehension extracts keys 1 and 2

Question 7

What makes this tuple declaration invalid:

my_tuple = (25)

A) Missing quotes

B) Requires comma for single element

C) Parentheses are incorrect

D) Numbers can't be in tuples

Correct Answer: B

Explanation:

- Tuples require commas to distinguish them from regular parentheses in mathematical operations.
- A single-element tuple **must** include a trailing comma: **(25,)**
- Without the comma, Python interprets **(25)** as an integer in parentheses rather than a tuple
- This behaviour is explicitly mentioned in Python's documentation:

pythonunitário = 'olá', *# Valid single-element tuple* vazio = () *# Valid empty tuple*

- Common pitfall: Developers often forget this syntax nuance when creating tuples with one element

---

Question 8

Which operation is NOT allowed on tuples?

A) Concatenation with +

B) Repetition with *

C) Item assignment

D) Slicing

Correct Answer: C

Explanation:

Tuples are immutable - once created, their elements cannot be modified through item assignment

- **Immutability** is the defining characteristic of tuples.
- Permitted operations:

pythont = (1, 2, 3) print(t + (4,)) *# Concatenation → (1,2,3,4)* print(t * 2) *# Repetition → (1,2,3,1,2,3)* print(t[1:]) *# Slicing → (2,3)*

- **Prohibited operations:**

pythont[0] = 5 *# Item assignment → TypeError* del t[1] *# Item deletion → TypeError*

- Why immutability matters:
    i. Hashability (can be dictionary keys)
    ii. Data integrity protection
    iii. Memory efficiency for static data
- Contrast with lists (mutable) that allow all modification operations

---

Question 9

What happens when you execute

my_list[2] = 15

if my_list = [10, 20, 30, 40]?

A) Creates new list

B) Modifies 3rd element

C) Throws IndexError

D) Adds new element

Correct Answer: B

Explanation:

- Lists are **mutable** in Python, meaning their elements can be modified in-place
- Index **2** refers to the **third element** (Python uses 0-based indexing):

python[10 (index 0), 20 (1), 30 (2), 40 (3)]

- The assignment **my_list = 15** replaces 30 with 15:

python[10, 20, 15, 40]

- Why other options are incorrect:
  - **A)** Doesn't create a new list - modifies existing one
  - **C)** No IndexError since index 2 exists (valid range: 0-3)
  - **D)** Doesn't add elements - requires append()/extend()

Question 10

What is the output of

print([1,2,3,4][-2:])

?

A) [3,4]

B) [2,3,4]

C) 4

D) [2,3]

Correct Answer: A

Explanation:

- Negative indices count backward from the end:

python[1 (index -4), 2 (-3), 3 (-2), 4 (-1)]

- **[-2:]** means "from index -2 to end":

pythonSlicing includes: [3, 4]

- Key slicing rules:
    - **start:end** includes **start** index, excludes **end**
    - Omitting end (**:**) means "go to end"
    - Works even when list length varies

Question 11

Which method combines two lists: list1 = [1,2], list2 = [3,4] → [1,2,3,4]?

A) list1.append(list2)

B) list1.extend(list2)

C) list1.add(list2)

D) list1.join(list2)

Correct Answer: B

Explanation:

- **B) extend()**: Merges elements from another iterable

pythonlist1.extend(list2) *# list1 becomes [1,2,3,4]*

- Why others fail:
    - **A) append()**: Adds list2 as single element → **[1,2,[3,4]]**
    - **C) add()**: Not a valid list method
    - **D) join()**: String method, converts list to string

Critical Distinction:

pythonextend(): list1 += list2 append(): list1 += [list2]

Question 12

Which is NOT valid empty list creation?

A) list()

B) []

C) list = list([])

D) list = {}

Correct Answer: D

Explanation:

- **D) {}** creates empty **dictionary**, not list
- Valid methods:

pythonlist() *# Constructor method* [] *# Literal syntax* list([]) *# Redundant but valid*

- Common pitfall:

pythonempty_dict = {} *# Dictionary* empty_list = [] *# List*

Type Verification:

pythonprint(type({})) *# <class 'dict'>* print(type([])) *# <class 'list'>*

Question 13

What is the output of

print(game_stats.get('wins', 0))

if

game_stats = {'losses': 5}

?

A) 5

B) KeyError

C) 0

D) None

Correct Answer: C

Explanation:

1. The **get()** method safely retrieves values from dictionaries without raising KeyErrors
2. Syntax: **dict.get(key, default_value)**
3. In the battle game's simulation code:

pythonPlayer1_wins = Player1_wins + 1 *# Similar to get() +1 pattern*

4. Breakdown:
    - **game_stats** contains **{'losses': 5}** (no 'wins' key)

    - **get('wins', 0)** returns default value 0 instead of error

Why Others Are Wrong:

- **A) 5**: Value for 'losses' key, not 'wins'
- **B) KeyError**: **dict['key']** would throw this, but **get()** prevents it
- **D) None**: Only returned if default isn't specified

Question 14

Which operation creates

{'attack': 70, 'defend': 30}

from

base = {'attack': 50}

?

A)

base.update(defend=30)

B)

base['attack'] += 20; base['defend'] = 30

C)

base.add({'attack':20, 'defend':30})

D)

base |= {'attack':20, 'defend':30}

Correct Answer: B

Explanation:

1. Direct key access modifies existing values/adds new keys
2. Mirrors AI decision weights in code:

python*# From computer_choice() function* weights = [0.7, 0.3] *# Similar key-value ratio*

3. Step-by-step execution:
    - **base['attack'] += 20** → 50 + 20 = 70
    - **base['defend'] = 30** → adds new key-value pair

Why Others Fail:

Option	Issue	Example from Game Code
A	**update()** requires proper syntax	**update({'defend':30})** would work
C	No **add()** method in dicts	Uses **append()/extend()** for lists
D	|=	merges dictionaries but overwrites values instead of modifying them numerically.

Question 15

What does

'defend' in battle_actions

check if

battle_actions = {'attack': True}

?

A) Value existence

B) Key existence

C) Key-value pair

D) Data type

Correct Answer: B

Explanation:

1. The **in** operator checks for **keys** in dictionaries
2. Used in AI decision-making logic:

pythonif Player1_defending or Player2_defending: *# Key check pattern* return 'attack'

3. Value checking requires explicit methods:

python'attack' in battle_actions.keys() *# Key check (redundant)* True in battle_actions.values() *# Value check*

Why Others Are Wrong:

Option	**Reality**	**Game Code Example**
A	Values require **.values()**	**damage = random.randint(10,20)** checks values
C	Requires both key & value	**items()** method needed for pairs
D	Type checking uses **type()**	**isinstance(battle_actions, dict)**