

P1.1. **\*\*Variables and Data Types\*\*** What is the correct way to declare and initialize a variable in Python?

- a) `int x = 5`      **b) `x = 5`**      c) `variable x = 5`      d) `declare x = 5`

P1.2. **\*\*Conditional Statements\*\*** What will be the output of the following code?

```
x = 10
```

```
if x > 5:
```

```
    print("Greater than 5")
```

```
else:
```

```
    print("Less than or equal to 5")
```

- a) Greater than 5**      b) Less than 5      c) Less than or equal to 5      d) Greater than or equal to 5

P1.3. **\*\*Loops\*\*** What will be the output of the following code?

```
for i in range(3):
```

```
    print(i)
```

- a) 0 1 2**      b) 1 2 3      c) 1 2      d) 0 1

P1.4. **\*\*Lists\*\*** Given a list `my_list = [1, 2, 3, 4]`, how can you add the element 5 to the end of the list?

- a) `my_list.add(5)`      **b) `my_list.append(5)`**      c) `my_list.insert(5)`      d) `my_list.extend(5)`

P1.5. **\*\*Functions\*\*** What is the purpose of the 'return' statement in a function?

- a) It prints a value to the console.      b) It defines the input parameters of the function.  
c) **It stops the execution of the function.**(also correct)      **d) It specifies the value to be sent back to the caller.**

P1.6. **\*\*Strings\*\*** Which of the following methods can be used to concatenate two strings in Python?

- a) `str1 + str2`**      b) `str1.concat(str2)`      c) `str1.concatenate(str2)`      d) `concat(str1, str2)`

P1.7. **\*\*Dictionaries\*\*** How do you access the value associated with the key 'age' in the dictionary 'person'?

```
person = {'name': 'John', 'age': 25, 'gender': 'Male'}
```

- a) `person['age']`**      b) `person.get('age')`      c) `person.value('age')`      d) `person(1)`

P1.8. **\*\*File Handling\*\*** What is the purpose of the `close()` method when working with files in Python?

- a) It closes the file, releasing its resources.**      b) It saves changes made to the file.  
c) It opens the file for reading.      d) It deletes the file.

P1.9. **\*\*List Slicing\*\*** Given a list `my_list = [1, 2, 3, 4, 5]`, what does `my_list[1:4]` return?

- a) `[1, 2, 3, 4]`      **b) `[2, 3, 4]`**      c) `[2, 3, 4, 5]`      d) `[1, 2, 3]`

P1.10. **\*\*Set Operations\*\*** What will be the result of the following set operation?

```
set1 = {1, 2, 3}
```

```
set2 = {3, 4, 5}
```

```
result = set1.intersection(set2)
```

- a) `{1, 2, 3, 4, 5}`      **b) `{3}`**      c) `{1, 2}`      d) `{4, 5}`

P1.11. **\*\*List Comprehension\*\*** What does the following list comprehension do?

```
squares = [x**2 for x in range(5)]
```

- a) Creates a list of squares from 0 to 5.      b) Creates a list of squares from 1 to 5.  
**c) Creates a list of squares from 0 to 4.**      d) Creates a list of cubes from 0 to 5.

P1.12. **\*\*Lambda Functions\*\*** What is the primary purpose of a lambda function in Python?

- a) To declare variables.      **b) To create anonymous functions.**  
c) To perform complex mathematical operations.      d) To define class methods.

P1.13. **\*\*List Slicing with Negative Indices\*\*** Consider the list `my_list = ['a', 'b', 'c', 'd', 'e']`. What does `my_list[-3:-1]` produce?

- a) **['c', 'd']**      b) ['b', 'c']      c) ['d', 'e']      d) ['c', 'e']

P1.14. **\*\*Slicing and Reversing\*\*** Given a list `elements = [10, 20, 30, 40, 50]`, what does `elements[::-1]` do?

- a) **Returns the list in reverse order: [50, 40, 30, 20, 10]**  
b) Returns the elements in their original order: [10, 20, 30, 40, 50]  
c) Returns an empty list: []  
d) Returns the last element only: [50]

P1.15. **\*\*Slicing with Omitted Indices\*\*** If `my_list = [1, 2, 3, 4, 5]`, what does `my_list[:3]` produce?

- a) **[1, 2, 3]**      b) [2, 3, 4]      c) [3, 4, 5]      d) [1, 2]
- 

P2.1. **\*\*List Comprehension\*\*** What does the following list comprehension do?

`squares = [x**2 for x in range(1, 6)]`

- a) Creates a list of squares from 1 to 6.      **b) Creates a list of squares from 1 to 5.**  
c) Creates a list of squares from 0 to 4.      d) Creates a list of cubes from 1 to 5.

P2.2. **\*\*List Comprehension with Condition\*\*** Given the list `numbers = [1, 2, 3, 4, 5]`, what does the following list comprehension produce?

`even_squares = [x**2 for x in numbers if x % 2 == 0]`

- a) **[4, 16]**      b) [1, 9, 25]      c) [2, 4]      d) [1, 3, 5]

P2.3. **\*\*Dictionary Comprehension\*\*** What does the following dictionary comprehension do?

`squares_dict = {x: x**2 for x in range(1, 4)}`

- a) **Creates a dictionary with keys from 1 to 3 and values as their squares.**  
b) Creates a dictionary with keys from 1 to 4 and values as their squares.  
c) Creates a dictionary with keys from 0 to 2 and values as their squares.  
d) Creates a dictionary with keys from 1 to 3 and values as cubes.

P2.4. **\*\*Nested List Comprehension\*\*** What does the following nested list comprehension produce?

`matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`

`flattened_matrix = [num for row in matrix for num in row]`

- a) **[1, 2, 3, 4, 5, 6, 7, 8, 9]**      b) [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  
c) [1, 4, 7, 2, 5, 8, 3, 6, 9]      d) [[1, 4, 7], [2, 5, 8], [3, 6, 9]]

P2.5. **\*\*Conditional Expression in Comprehension\*\*** What does the following list comprehension do?

`result = ['Even' if x % 2 == 0 else 'Odd' for x in range(1, 6)]`

- a) **Creates a list of strings indicating whether each number from 1 to 5 is even or odd.**  
b) Creates a list of even numbers from 1 to 5.  
c) Creates a list of integers indicating whether each number from 1 to 5 is even or odd.  
d) Creates a list of strings containing the numbers from 1 to 5.

C1.1. **\*\*Arrays\*\*** How do you access the third element of an array named `numbers` in C? - **Wrong question, marks will be given if attempted**

- a) **numbers[3]**      b) numbers(3)      c) numbers.third      d) numbers.3      -- correct answer -> numbers[2]

C1.2. **\*\*Functions\*\*** In C, what is the purpose of the `return` statement in a function?

- a) It prints a value to the console.      **b) It stops the execution of the function.** (also correct)  
**c) It specifies the value to be sent back to the caller.**      d) It defines the input parameters of the function.

C1.3. **\*\*Pointers\*\*** What is the role of the \* symbol when working with pointers in C?

- a) It represents the address of a variable.
- b) **It is used to declare a pointer variable.** (also correct)
- c) It is used for multiplication.
- d) **It dereferences a pointer, accessing the value it points to.**

C1.4. **\*\*Structures\*\*** How do you declare a structure named *Person* with members *name* and *age* in C?

- a) **struct Person { char name; int age; };**
- b) struct { char name; int age; } Person;
- c) struct { Person char; age int; };
- d) struct Person { name char; age int; };

C1.5. **\*\*File Handling\*\*** Which function is used to open a file in C?

- a) open()
  - b) file\_open()
  - c) **fopen()**
  - d) read\_file()
- 

C2.1. **\*\*Dynamic Memory Allocation\*\*** In C, which function is used to dynamically allocate and deallocate memory?

- a) **malloc() and free()**
- b) allocate() and deallocate()
- c) malloc() and realloc()
- d) alloc() and release()

C2.2. **\*\*Strings\*\*** From *string.h*, which functions is used to concatenate two strings, and which function is used to compare two strings?

- a) **strcat() and strcmp()**
- b) concat() and strcmp()
- c) merge() and compare()
- d) append() and equals()

C2.3. **\*\*Pointer Arithmetic\*\*** Given the following code snippet:

```
int numbers[] = {1, 2, 3, 4, 5};  
int *ptr = numbers;  
printf("%d", *(ptr + 2));
```

What will be printed to the console?

- a) 1
- b) 2
- c) **3**
- d) 4

C2.4. **\*\*Array of Pointers\*\*** Given the declaration *int \*arr[5];*, what does this signify in C?

- a) It declares an array of 5 integers.
- b) It declares a pointer to an array of 5 integers.
- c) **It declares an array of 5 pointers to integers.**
- d) It declares an array of 5 double pointers.

C2.5. **\*\*Dynamic Memory Allocation\*\*** What is the function used to deallocate memory allocated by malloc?

- a) dealloc
- b) **free**
- c) dispose
- d) release

C2.6. **\*\*Void Pointer\*\*** What is the purpose of a void pointer (void \*) in C?

- a) It cannot be used in C.
- b) **It is a pointer that can point to any data type.**
- c) It is a pointer specifically for character data.
- d) It is a constant pointer.

C2.7. **\*\*Accessing Structure Members\*\*** Given the structure definition:

```
struct Student {  
    char name[50];  
    int age;  
    float gpa;  
};
```

How do you access the age member of a structure variable named stud?

- a) stud->age
- b) stud::age
- c) **stud.age**
- d) stud[1].age

C2.8. **\*\*Sizeof and Pointers\*\*** In C, what does the sizeof operator return when used with a pointer?

- a) The size of the data type pointed to by the pointer.
- b) **The size of the pointer itself.**
- c) The total size of the allocated memory block.
- d) The number of elements in the array pointed to by the pointer.

C2.9: **\*\*Sizeof and Struct\*\*** Consider the following C code:

```
struct Point { int x; int y; };  
long int size = sizeof(struct Point);
```

What does the variable *size* represent?

- a) The number of members in the struct.
- b) The total size of the struct in bytes.**
- c) The size of each member in the struct.
- d) The number of bytes occupied by the data type of the struct.

C2.10: **\*\*Undefined Behavior\*\*** Which of the following situations may lead to undefined behavior in C?

- a) Accessing an array element using a negative index.**
- b) Performing arithmetic operations on void pointers.**
- c) Calling a function without a prototype.**
- d) Using an uninitialized variable.**

B1. **\*\*Variable Declaration and Initialization\*\*** Consider the following C code:

```
int x; x = 5;
```

Which of the following statements is true?

- a) The variable x is declared and initialized at the same time.
- b) The variable x is declared but not initialized.**
- c) The variable x is initialized but not declared.
- d) The code will result in a compilation error.

B2. **\*\*Variable Scope\*\*** Consider the following C code:

```
int x = 5;  
if (1) {  
    int x = 10;  
    printf("%d ", x);  
}  
printf("%d", x);
```

What will be printed?

- a) 10 10**
- b) 5 5
- c) 10 5
- d) 5 10

B3. **\*\*Loops\*\*** Which loop in C is best suited for situations where the number of iterations is known before entering the loop?

- a) while
- b) do-while
- c) for**
- d) if

B4. **\*\*Pointer Arithmetic\*\*** If *ptr* is a pointer to an integer in C, what does *ptr + 3* represent?

- a) The value stored at the memory location three positions after *ptr*.
- b) The memory address three positions after the memory address pointed to by *ptr*.
- c) The third integer value after the one pointed to by *ptr*.**
- d) An error, as pointer arithmetic with integers is not allowed.

B5. **\*\*Array and Pointers\*\*** In C, how is the name of an array related to a pointer?

- a) The array name is a constant pointer to the first element of the array.**
- b) The array name is a pointer to the last element of the array.
- c) The array name is a pointer to the middle element of the array.
- d) The array name is not related to pointers.

B6. **\*\*Nested Loops\*\*** Consider the following nested loop in C:

```
for (int i = 1; i <= 3; i++) {  
    for (int j = 1; j <= 2; j++) {  
        printf("%d ", i * j);  
    }  
}
```

What will be the output?

a) 1 2 3 4 5 6      b) 1 2 1 2 1 2      c) 2 4 6 8 10 12      d) 1 1 2 2 3 3      -- correct answer -> 1 2 2 4 3 6

B7. **\*\*do-while Loop Evaluation\*\*** What is the key characteristic of a do-while loop compared to a while loop?

- a) The do-while loop must have an explicit counter.
- b) The loop body of a do-while is always executed at least once.**
- c) The do-while loop can only be used for infinite loops.
- d) The do-while loop cannot contain conditional statements.

B8. **\*\*Ternary Operator\*\*** What is the correct syntax for the ternary operator in C?

- a) condition ? expression\_if\_true : expression\_if\_false**
- b) expression\_if\_true ? condition : expression\_if\_false
- c) expression\_if\_true : condition ? expression\_if\_false
- d) condition : expression\_if\_false ? expression\_if\_true

B9. **\*\*Multidimensional Array Access\*\*** Given the following declaration:

```
int matrix[3][4] = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 } };
```

How do you access the value 7 in the matrix array?

- a) matrix[1][2]**
- b) matrix[2][1]
- c) matrix[3][2]
- d) matrix[2][3]

B10. **\*\*Switch Case\*\*** In a switch statement, what is the behaviour when there is no break statement between cases?

- a) It results in a syntax error.
- b) It is allowed, and control falls through to the next case.**
- c) It automatically adds a break statement.
- d) It skips the current case and moves to the default case.

B11. **\*\*Variable Scope\*\*** In C, what is the scope of a variable declared inside a function (but not as a parameter)?

- a) Global scope
- b) Local scope**
- c) Static scope
- d) Dynamic scope

B12. **\*\*Pointer Arithmetic with Arrays\*\*** Consider the following statements:

I. Adding an integer n to a pointer increases the address it points to by n \* sizeof(element\_type).

II. Subtracting an integer n from a pointer decreases the address it points to by n \* sizeof(element\_type).

III. Subtracting one pointer from another yields the number of elements between them.

Which of the above statements is/are true?

- a) I only
- b) II only
- c) III only
- d) I, II, and III**

B13. **\*\*Recursive Calculation\*\*** Consider the following recursive function on an integer number:

```
int fact(int n) {  
    if (n == 0 || n == 1)    return 1;  
    else                    return n + fact(n - 1);  
}
```

What will be the result of fact(5)?

- a) 15**
- b) 120
- c) 25
- d) 720

B14. **\*\*Sizeof Operator\*\*** Consider the following C code:

```
int arr[5];
```

```
int size = sizeof(arr);
```

What does the variable size represent?

- a) The number of elements in the array.
- b) The total size of the array in bytes.**
- c) The size of each element in the array.
- d) The number of bytes occupied by the data type of the array.

B15. **\*\*Initializing 2D Arrays\*\*** What is the correct syntax for initializing a 2D array in C?

a) `int matrix[][] = { { 1, 2 }, { 3, 4 } };`

b) `int matrix[2][2] = { 1, 2, 3, 4 };`

**c) `int matrix[2][2] = {{1, 2}, {3, 4}};`**

d) `int matrix[2][2] = {[1][1]=1, [1][2]=2, [2][1]=3, [2][2]=4};`

---