

Achieve your Python Certified Associate Programmer (PCAP-31-3) certification with confidence using the PCAP-31-3 Preparation Book.

This expertly crafted guide is tailored to equip you with the knowledge and skills required to pass the certification exam and advance your career in the dynamic field of Python programming.

Key Features:

- **Detailed Explanations:** Benefit from clear, concise explanations and insights that help you understand not just the "how," but also the "why" behind each concept.
- **Mock Exams:** Assess your readiness with full-length mock exams that simulate the actual PCAP-31-3 testing environment, providing you with a realistic preview of what to expect on exam day.



GROUP
XPERT
ISBN: 978-3-910818-08-8



Master the Python Certified Associate Programmer Exam with Confidence

```
def dotwrite(sinh):  
    database = getDatabase()  
    label=register(sys.argv[1])  
    print(''.join([label[i:i+1] for i in range(0, len(label), 1)]))  
    if not(sinh).strip():  
        print('' + sinh)  
    else:  
        print('' + sinh)
```

LICENSED ONLY FOR: WBS TRAINING AG
(PYDASC71E-13)

Preparation of PCAP-31-03

1. Auflage

E-Book-ISBN: 978-3-910818-08-8

Dieses Werk ist urheberrechtlich geschützt.

Alle Rechte (Übersetzung, Nachdruck und Vervielfältigung) vorbehalten. Kein Teil des Werks darf ohne schriftliche Genehmigung des Autors in irgendeiner Form – auch nicht für Zwecke der Unterrichtsgestaltung reproduziert, verarbeitet, vervielfältigt oder verbreitet werden.

Dieses Buch wurde mit größter Sorgfalt erstellt, ungeachtet dessen kann Autor, Herausgeber oder Übersetzer für mögliche Fehler und deren Folgen eine juristische Verantwortung oder irgendeine Haftung übernehmen. Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären.

TABLE OF CONTENT

Part 1: Fundamentals of Python Programming.....	
1. Basics (30x Questions)	4
2. Control-Flow (20x Questions)	17
3. Data Structures (60x Questions)	28
4. Input and type casting (20x Questions)	53
5. For Loop (30x Questions)	62
6. While Loop (20x Questions)	77
 Part 2: Advanced Topics.....	
1. Comprehensions (30x Questions)	90
2. Exceptions (75x Questions)	102
3. Modules (20x Questions)	140
4. Built-In Functions (80x Questions)	147
 Part 3: PCAP-based Preparation	
1. Modules and Packages (50x Questions)	179
2. Exceptions (50x Questions)	191
3. Strings (50x Questions).....	205
4. Object Oriented Programming (OOP) (50x Questions)	217
5. Miscellaneous (50x Questions)	240
 Part 4: Complete Exams	
Exam 1 (40x Questions)	258
Exam 2 (40x Questions)	275
Exam 3 (40x Questions)	293
Exam 4 (40x Questions)	309

1.1 Basics

Required Pre-Knowledge

- Basic Python syntax and execution.
 - Understanding of Python data types: `int`, `float`, `str`, `bool`, `list`, `tuple`, `set`, `dict`, `complex`.
 - Understanding of immutable vs mutable types in Python.
 - Familiarity with built-in functions: `len()`, `type()`, `sorted()`, `max()`, `min()`, `abs()`, `bool()`, `get()`.
 - Knowledge of Python mathematical operators: `+`, `**`, `%`, `//`.
 - Logical operators in Python: `and`, `or`, `not`, and their precedence.
 - Identifying and fixing syntax errors in Python code.
 - Understanding Python functions and their syntax.
 - Understanding tuple packing.
-

Quiz

It contains 30 questions. Please answer the questions on separate paper or in an Excel file, and then compare your answers with those in the answers section after the quiz.

Q1

What is the output of the following code?

```
x = 5
y = x + 2
print(y)
```

Options:

- 5
 - 7
 - 52
 - Error
-

Q2

What is the syntax error in the following code?

```
def greet()
    print("Hello, World!")
```

Options:

- Missing colon after `print`
 - Missing parentheses in `print`
 - Missing colon after `def greet`
 - No error
-

❓ Q3

What is the syntax error in the following code?

```
print "Hello, World!"
```

Options:

- Missing colon
 - Missing parentheses
 - Incorrect indentation
 - No error
-

❓ Q4

What is the syntax error in the following code?

```
if True:  
    print("True")
```

Options:

- Missing colon after `if True`
 - Missing parentheses in `print`
 - Incorrect indentation
 - No error
-

❓ Q5

What is the output of the following code?

```
value = 3.14  
print(type(value))
```

Options:

- `<class 'int'>`

- <class 'float'>
 - <class 'str'>
 - <class 'bool'>
-

❓ Q6

What is the output of the following code?

```
print(bool(0), bool(1))
```

Options:

- True True
 - False True
 - True False
 - False False
-

❓ Q7

What is the output of the following code?

```
z = 2 + 3j
print(z.real, z.imag)
```

Options:

- 2 3
 - 3 2
 - (2, 3)
 - 2+3j
-

❓ Q8

Which of the following statements is true?

Options:

- Lists are immutable, tuples are mutable
 - Both lists and tuples are mutable
 - Lists are mutable, tuples are immutable
 - Both lists and tuples are immutable
-

❓ Q9

Which of the following is true about a **set** in Python?

Options:

- Sets are ordered
 - Sets can contain duplicate elements
 - Sets are mutable
 - Sets are immutable
-

❓ Q10

What is the output of the following code?

```
my_dict = {'a': 1, 'b': 2}  
print(my_dict.get('c', 3))
```

Options:

- 1
- 2
- 3
- None

❓ Q11

What is the output of the following code?

```
person = {'name': 'Alice', 'age': 25}  
print(person['name'])
```

Options:

- 'Alice'
 - 'age'
 - 25
 - Error
-

❓ Q12

What is the output of the following code?

```
t = 1, 2, 3  
print(type(t))
```

Options:

- `<class 'list'>`
 - `<class 'tuple'>`
 - `<class 'dict'>`
 - `<class 'set'>`
-

❓ Q13

What is the output of the following code?

```
numbers = [1, 2, 2, 3, 4, 4, 5]
unique_numbers = set(numbers)
print(unique_numbers)
```

Options:

- `[1, 2, 3, 4, 5]`
 - `{1, 2, 3, 4, 5}`
 - `[(1, 2, 3, 4, 5)]`
 - Error
-

❓ Q14

What is the output or result of the following code?

```
my_tuple = (1, 2, 3)
my_tuple[0] = 4
```

Options:

- The tuple becomes `(4, 2, 3)`
 - A new tuple `(4, 2, 3)` is created
 - The operation is allowed without errors
 - Raises a `TypeError`
-

❓ Q15

What will be the result of executing the following code?

```
fs = frozenset([1, 2, 3])
fs.add(4)
```

Options:

- `frozenset({1, 2, 3, 4})`
 - `frozenset({1, 2, 3})`
 - Raises an `AttributeError`
 - Raises a `TypeError`
-

❓ Q16

What will be the result of the following code?

```
s = "Hello"  
s[0] = "h"
```

Options:

- `hello`
 - `Hello`
 - Raises a `TypeError`
 - None
-

❓ Q17

What is the output of the following code?

```
my_list = [10, 20, 30, 40]  
print(len(my_list))
```

Options:

- 3
 - 4
 - 5
 - Error
-

❓ Q18

What is the output of the following code?

```
print(type([1, 2, 3]), type({1: 'a', 2: 'b'}))
```

Options:

- <class 'list'> <class 'dict'>
 - <class 'tuple'> <class 'dict'>
 - <class 'list'> <class 'list'>
 - <class 'dict'> <class 'list'>
-

?

Q19

What is the output of the following code?

```
numbers = [3, 1, 4, 1, 5]
print(sorted(numbers))
```

Options:

- [3, 1, 4, 1, 5]
 - [1, 1, 3, 4, 5]
 - [5, 4, 3, 1, 1]
 - Error
-

?

Q20

What is the output of the following code?

```
nums = [10, 20, 30, 40]
print(max(nums) - min(nums))
```

Options:

- 30
 - 40
 - 10
 - 20
-

?

Q21

What is the output of the following code?

```
print(abs(-10.5))
```

Options:

- -10.5

- 10.5
 - 0
 - Error
-

❓ Q22

What is the output of the following code?

```
letters = ['b', 'a', 'd', 'c']
print(sorted(letters, reverse=True))
```

Options:

- ['a', 'b', 'c', 'd']
 - ['d', 'c', 'b', 'a']
 - ['b', 'a', 'd', 'c']
 - Error
-

❓ Q23

What is the output of the following code?

```
result = 2 ** 3
print(result)
```

Options:

- 6
 - 8
 - 9
 - 5
-

❓ Q24

What is the output of the following code?

```
print(10 % 3)
```

Options:

- 1
- 2

- 3
 - 0
-

❓ Q25

What is the output of the following code?

```
print(7 // 2)
```

Options:

- 3.5
 - 4
 - 3
 - Error
-

❓ Q26

What is the output of the following code?

```
a = True
b = False
print(a and b)
```

Options:

- True
 - False
 - Error
 - None
-

❓ Q27

What is the output of the following code?

```
x = 0
y = "Hello"
print(x or y)
```

Options:

- 0
- "Hello"

- True
 - False
-

❓ Q28

What is the output of the following code?

```
print(not False)
```

Options:

- True
 - False
 - Error
 - None
-

❓ Q29

What is the output of the following code?

```
print(True or False and False)
```

Options:

- True
 - False
 - Error
 - None
-

❓ Q30

What is the output of the following code?

```
a = False
b = True
c = False
print(a or b and c)
```

Options:

- True
- False

- Error
 - None
-

Licensed for: WBS Training AG
PYDASC71e-13

✓ The Answers

Question Nr.	Answer	Explanation
1	B	The variable <code>x</code> is assigned the value <code>5</code> . The expression <code>x + 2</code> adds <code>5</code> and <code>2</code> , resulting in <code>7</code> , which is assigned to <code>y</code> . Therefore, <code>print(y)</code> outputs <code>7</code> .
2	C	Function definitions in Python require a colon at the end of the <code>def</code> statement. The correct syntax is <code>def greet():</code> .
3	B	In Python 3, <code>print</code> is a function and requires parentheses. The correct syntax is <code>print("Hello, World!")</code> .
4	C	In Python, the block under an <code>if</code> statement must be indented. <code>print("True")</code> should be indented to indicate it belongs to the <code>if</code> block.
5	B	The variable <code>value</code> is assigned a floating-point number <code>3.14</code> . The <code>type()</code> function returns <code><class 'float'></code> , indicating that <code>value</code> is of type <code>float</code> .
6	B	In Python, <code>bool(0)</code> is <code>False</code> and <code>bool(1)</code> is <code>True</code> . Therefore, the output is <code>False True</code> .
7	A	For a complex number <code>z = 2 + 3j</code> , <code>z.real</code> is <code>2</code> and <code>z.imag</code> is <code>3</code> . Thus, <code>print(z.real, z.imag)</code> outputs <code>2 3</code> .
8	C	Lists in Python are mutable, meaning their contents can be changed. Tuples are immutable and cannot be altered after creation.
9	C	Sets in Python are unordered collections of unique elements and are mutable, meaning you can add or remove items.
10	C	The <code>get()</code> method returns the value for the specified key if the key is in the dictionary. If not, it returns the default value provided, which is <code>3</code> .
11	A	Accessing <code>person['name']</code> retrieves the value associated with the key <code>'name'</code> , which is <code>'Alice'</code> .
12	B	The syntax <code>t = 1, 2, 3</code> creates a tuple. Therefore, <code>type(t)</code> is <code><class 'tuple'></code> .
13	B	Converting a list to a <code>set</code> removes duplicate elements. The output is <code>{1, 2, 3, 4, 5}</code> .
14	D	Tuples in Python are immutable, meaning their elements cannot be changed after creation. Attempting to assign <code>4</code> to <code>my_tuple[0]</code> raises a <code>TypeError</code> .
15	C	<code>frozenset</code> objects are immutable and do not have an <code>add</code> method. Attempting to call <code>fs.add(4)</code> raises an <code>AttributeError</code> .
16	C	Strings in Python are immutable. Attempting to change a character in a string will raise a <code>TypeError</code> .
17	B	The list <code>my_list</code> contains four elements. The <code>len()</code> function returns the number of items in the list, which is <code>4</code> .

Question Nr.	Answer	Explanation
18	A	[1, 2, 3] is a list, and {1: 'a', 2: 'b'} is a dictionary. Therefore, type([1, 2, 3]) is <class 'list'> and type({1: 'a', 2: 'b'}) is <class 'dict'>.
19	B	The sorted() function returns a new list containing all items from the iterable in ascending order. Thus, [1, 1, 3, 4, 5].
20	A	max(nums) is 40 and min(nums) is 10. Therefore, 40 - 10 equals 30.
21	B	The abs() function returns the absolute value of a number. abs(-10.5) is 10.5.
22	B	The sorted() function with reverse=True sorts the list in descending order, resulting in ['d', 'c', 'b', 'a'].
23	B	The ** operator performs exponentiation. 2 ** 3 equals 8.
24	A	10 % 3 equals 1.
25	C	The // operator performs integer (floor) division. 7 // 2 equals 3.
26	B	The and operator returns True only if both operands are True. Since b is False, a and b evaluates to False.
27	B	The or operator returns the first truthy value. 0 is falsy, so it returns "Hello".
28	A	The not operator negates the boolean value. not False is True.
29	A	According to operator precedence, and is evaluated before or. So, False and False is False, then True or False is True.
30	B	Operator precedence dictates that and is evaluated before or. So, b and c is True and False, which is False. Then, a or False is False or False, which is False.

1.2 Control-Flow

Required Pre-Knowledge

- Understanding of `if`, `elif`, `else` statements in Python.
- Familiarity with the `match` statement and pattern matching in Python.
- Knowledge of logical operators: `and`, `or`, `not`, and their precedence.
- Ability to identify and fix syntax errors related to indentation and statement structure.
- Basic Python syntax and execution flow.
- Understanding of basic Python data types: `int`, `str`.
- Understanding of nested conditional statements.
- Familiarity with multi-pattern matching in `match` statements.

Quiz

It contains 20 questions. Please answer the questions on separate paper or in an Excel file, and then compare your answers with those in the answers section after the quiz.

Q1

What is the output of the following code?

```
x = 10
if x > 5:
    print("Greater than 5")
```

Options:

- Greater than 10
- Greater than or equal to 5
- Greater than 5
- No output

Q2

What is the output of the following code?

```
x = 4
if x > 5:
    print("Greater than 5")
```

Options:

- Greater than 5
 - Less than or equal to 5
 - No output
 - Error
-

?

Q3

What is the output of the following code?

```
x = 4
if x > 2 and x < 5:
    print("Between 2 and 5")
```

Options:

- Greater than 5
 - Between 2 and 5
 - Less than 2
 - No output
-

?

Q4

What is the output of the following code?

```
x = 7
if x > 5:
    print("First condition met")
if x > 6:
    print("Second condition met")
```

Options:

- First condition met
 - Second condition met
 - First condition met
Second condition met
 - No output
-

?

Q5

What is the output of the following code?

```
x = 10
```

```
if x > 5:  
    print("First if")  
if x > 8:  
    print("Second if")
```

Options:

- First if
 - Second if
 - First if
Second if
 - No output
-

❓ Q6

What is the output of the following code?

```
x = 3  
if x > 5:  
    print("Greater than 5")  
elif x > 2:  
    print("Greater than 2 but not greater than 5")  
else:  
    print("2 or less")
```

Options:

- Greater than 5
 - Greater than 2 but not greater than 5
 - 2 or less
 - No output
-

❓ Q7

What is the output of the following code?

```
x = 1  
if x > 5:  
    print("Greater than 5")  
elif x > 2:  
    print("Greater than 2 but not greater than 5")  
else:  
    print("2 or less")
```

Options:

- Greater than 5
 - Greater than 2 but not greater than 5
 - 2 or less
 - No output
-

❓ Q8

What is the output of the following code?

```
x = 6
if x > 5:
    print("Greater than 5")
elif x > 3:
    print("Greater than 3")
else:
    print("3 or less")
```

Options:

- Greater than 5
 - Greater than 3
 - 3 or less
 - No output
-

❓ Q9

What is the output of the following code?

```
x = 5
if x > 5:
    print("Greater than 5")
elif x == 5:
    print("Equal to 5")
elif x > 3:
    print("Greater than 3")
else:
    print("3 or less")
```

Options:

- Greater than 5
 - Equal to 5
 - Greater than 3
 - 3 or less
-

?

Q10

What is the output of the following code?

```
x = 12
if x > 10:
    if x % 2 == 0:
        print("Greater than 10 and even")
    else:
        print("Greater than 10 and odd")
else:
    print("10 or less")
```

Options:

- Greater than 10 and even
 - Greater than 10 and odd
 - 10 or less
 - No output
-

?

Q11

What is the output of the following code?

```
command = "exit"
match command:
    case "start":
        print("Starting")
    case "stop":
        print("Stopping")
```

Options:

- Starting
 - Stopping
 - No output
 - Error
-

?

Q12

What is the output of the following code?

```
command = "restart"
match command:
    case "restart":
```

```
print("Restarting")
case "start" | "restart":
    print("Starting or Restarting")
```

Options:

- Restarting
 - Starting or Restarting
 - No output
 - Error
-

❓ Q13

What is the output of the following code?

```
command = "start"
match command:
    case "start":
        print("Starting")
    case "stop":
        print("Stopping")
```

Options:

- Starting
 - Stopping
 - No output
 - Error
-

❓ Q14

What is the output of the following code?

```
x = 7
if x > 10:
    print("Greater than 10")
else:
    if x > 5:
        print("Greater than 5 but not greater than 10")
    else:
        print("5 or less")
```

Options:

- Greater than 10

- Greater than 5 but not greater than 10
 - 5 or less
 - No output
-

❓ Q15

What is the output of the following code?

```
status = "success"
match status:
    case "success":
        print("Operation was successful")
    case "failure":
        print("Operation failed")
```

Options:

- Operation was successful
 - Operation failed
 - No output
 - Error
-

❓ Q16

What is the output of the following code?

```
command = "exit"
match command:
    case "start" | "restart":
        print("Starting or Restarting")
    case "exit":
        print("Exiting")
```

Options:

- Starting or Restarting
 - Exiting
 - No output
 - Error
-

❓ Q17

What is the output of the following code?

```
command = "unknown"
match command:
    case "start":
        print("Starting")
    case "stop":
        print("Stopping")
    case _:
        print("Unknown command")
```

Options:

- Starting
 - Stopping
 - Unknown command
 - No output
-

❓ Q18

What is the output of the following code?

```
x = 4
if x > 2 and x < 5:
    print("Between 2 and 5")
```

Options:

- Greater than 5
 - Between 2 and 5
 - Less than 2
 - No output
-

❓ Q19

What is the output of the following code?

```
x = 5
if x > 5:
    print("Greater than 5")
elif x == 5:
    print("Equal to 5")
elif x > 3:
    print("Greater than 3")
else:
    print("3 or less")
```

Options:

- Greater than 5
 - Equal to 5
 - Greater than 3
 - 3 or less
-

❓ Q20

What is the output of the following code?

```
command = "pause"
match command:
    case "start":
        print("Starting")
    case "stop":
        print("Stopping")
```

Options:

- Starting
- Stopping
- No output
- Error

✓ The Answers

Question Nr.	Answer	Explanation
1	C	The variable <code>x</code> is set to <code>10</code> . The condition <code>x > 5</code> is <code>True</code> , so "Greater than 5" is printed.
2	C	The condition <code>x > 5</code> is <code>False</code> when <code>x = 4</code> , so nothing is printed.
3	B	<code>x</code> is <code>4</code> , which satisfies both <code>x > 2</code> and <code>x < 5</code> . Hence, "Between 2 and 5" is printed.
4	C	Both conditions <code>x > 5</code> and <code>x > 6</code> are <code>True</code> for <code>x = 7</code> , so both print statements are executed.
5	C	Both <code>x > 5</code> and <code>x > 8</code> are <code>True</code> for <code>x = 10</code> , so both print statements are executed.
6	B	<code>x</code> is <code>3</code> , which is not greater than <code>5</code> but is greater than <code>2</code> . Hence, the <code>elif</code> block is executed.
7	C	<code>x</code> is <code>1</code> , which does not satisfy any of the first two conditions. The <code>else</code> block is executed, printing "2 or less".
8	A	<code>x</code> is <code>6</code> , which satisfies the first condition <code>x > 5</code> . The subsequent <code>elif</code> and <code>else</code> blocks are not evaluated.
9	B	<code>x</code> is <code>5</code> , which satisfies the second condition <code>x == 5</code> . Only the first matching <code>elif</code> block is executed.
10	A	<code>x</code> is <code>12</code> , which is greater than <code>10</code> and even (<code>12 % 2 == 0</code>), so the first nested print statement is executed.
11	C	<code>command</code> is "exit", which doesn't match any case. Since there's no default case, nothing is printed.
12	A	The first case "restart" matches and is executed. The second case is not evaluated due to the first match.
13	A	<code>command</code> is "start", which matches the first case and prints "Starting".
14	B	<code>x</code> is <code>7</code> , which is not greater than <code>10</code> . The nested <code>if</code> checks <code>x > 5</code> , which is <code>True</code> , so "Greater than 5 but not greater than 10" is printed.
15	A	<code>status</code> is "success", which matches the first case and prints "Operation was successful".
16	B	The <code>command</code> "exit" matches the second case, printing "Exiting".
17	C	<code>command</code> is "unknown", which doesn't match any specific case. The default case <code>_</code> is executed, printing "Unknown command".
18	B	<code>x</code> is <code>4</code> , which satisfies both <code>x > 2</code> and <code>x < 5</code> . Hence, "Between 2 and 5" is printed.

Question Nr.	Answer	Explanation
19	B	<code>x</code> is 5, which satisfies the second condition <code>x == 5</code> . Only the first matching <code>elif</code> block is executed.
20	C	<code>command</code> is "pause", which doesn't match any of the specified cases. Since there's no default case, nothing is printed.

Licensed for: WBS Training AG
PYDASC71e-13

1.3 Data Structures

Required Pre-Knowledge

- **Basic Python Data Types:** Understanding `str`, `list`, `tuple`, `dict`, `set`, and `frozenset`.
 - **String Methods:** Familiarity with methods such as `upper()`, `lower()`, `replace()`, `split()`, `startswith()`, `endswith()`, `find()`, `format()`, and `strip()`.
 - **List Operations:** Proficiency with methods like `append()`, `remove()`, `extend()`, `pop()`, `insert()`, `sort()`, `count()`, `reverse()`, `index()`, `clear()`, and `copy()`.
 - **Tuple Characteristics:** Understanding that tuples are immutable and knowing how to use `count()`, `index()`, and slicing on tuples.
 - **Dictionary Operations:** Knowledge of methods like `get()`, `keys()`, `values()`, `items()`, `update()`, `pop()`, `popitem()`, `clear()`, `setdefault()`, and merging dictionaries using `**`.
 - **Set Methods:** Familiarity with `add()`, `remove()`, `discard()`, `union()`, `intersection()`, `difference()`, `symmetric_difference()`, `difference_update()`, `update()`, `issubset()`, and `copy()`.
 - **Frozenset Properties:** Understanding frozenset immutability and methods such as `difference()`, `issubset()`, and `union()`.
 - **Combining Iterables:** Using `zip()` to pair keys with values or to create tuples from multiple iterables.
 - **Indexing and Slicing:** General comprehension of accessing elements by index and slicing sequences.
-

Quiz

It contains 60 questions. Please answer the questions on separate paper or in an Excel file, and then compare your answers with those in the answers section after the quiz.

Q1

What is the output of the following code?

```
text = "hello world"
print(text.upper())
```

Options:

- HELLO WORLD
 - Hello World
 - hello world
 - HeLLo WoRLd
-

?

Q2

What is the output of the following code?

```
words = ['Hello', 'World']
sentence = ' '.join(words)
print(sentence)
```

Options:

- HelloWorld
 - Hello World
 - ['Hello', 'World']
 - Hello, World
-

?

Q3

What is the output of the following code?

```
words = ['Python', 'is', 'fun']
sentence = ' '.join([word.upper() for word in words])
print(sentence)
```

Options:

- Python is fun
 - PYTHON IS FUN
 - Python Is Fun
 - ['PYTHON', 'IS', 'FUN']
-

?

Q4

What is the output of the following code?

```
text = "banana"
new_text = text.replace("a", "o")
print(new_text)
```

Options:

- bonono
- banana
- bnnn

- bonana
-

❓ Q5

What is the output of the following code?

```
chars = ['P', 'y', 't', 'h', 'o', 'n']
word = ''.join(chars)
print(word)
```

Options:

- P y t h o n
 - Python
 - ['P', 'y', 't', 'h', 'o', 'n']
 - Pyth o n
-

❓ Q6

What is the output of the following code?

```
text = "Python is awesome"
words = text.split()
print(words)
```

Options:

- Python is awesome
 - ['Python', 'is', 'awesome']
 - ['Python is awesome']
 - ['Python', 'isawesome']
-

❓ Q7

What is the output of the following code?

```
name = "Lily"
age = 22
sentence = "{} is {} years old".format(name, age)
print(sentence)
```

Options:

- Lily is 22 years old
 - {} is {} years old
 - Lily is twenty-two years old
 - Lily 22 years old
-

❓ Q8

What is the output of the following code?

```
text = "OpenAI ChatGPT"  
print(text.startswith("Open"))
```

Options:

- True
 - False
 - "True"
 - "False"
-

❓ Q9

What is the output of the following code?

```
text = "Data Science"  
position = text.find("Science")  
print(position)
```

Options:

- 5
 - 4
 - -1
 - 6
-

❓ Q10

What is the output of the following code?

```
text = "HELLO WORLD"  
print(text.lower())
```

Options:

- HELLO WORLD
 - hello world
 - Hello World
 - hELLO wORLD
-

❓ Q11

What is the output of the following code?

```
text = "banana"
new_text = text.replace("a", "o", 2)
print(new_text)
```

Options:

- bonono
 - bonana
 - banana
 - bonona
-

❓ Q12

What is the output of the following code?

```
text = "OpenAI ChatGPT"
print(text.endswith("ChatGPT"))
```

Options:

- True
 - False
 - "True"
 - "False"
-

❓ Q13

What is the output of the following code?

```
text = "Hello World "
clean_text = text.strip()
print(clean_text)
```

Options:

- "Hello World"
 - "Hello World"
 - "Hello World "
 - "HelloWorld"
-

❓ Q14

What is the output?

```
fruits = ['apple', 'banana']
fruits.append('cherry')
print(fruits)
```

Options:

- ['apple', 'banana', 'cherry']
 - ['apple', 'banana']
 - ['apple', 'banana', 'cherry', 'cherry']
 - ['cherry', 'apple', 'banana']
-

❓ Q15

What is the output?

```
numbers = [1, 2, 3, 2]
numbers.remove(2)
print(numbers)
```

Options:

- [1, 3, 2]
 - [1, 2, 3, 2]
 - [1, 3]
 - [1, 2, 2]
-

❓ Q16

What is the output?

```
numbers = [1, 2, 3, 4]
print(numbers.pop(2))
print(numbers)
```

Options:

- 3 [1, 2, 4]
 - 2 [1, 3, 4]
 - 4 [1, 2, 3]
 - 3 [1, 2, 3, 4]
-

❓ Q17

What is the output?

```
list1 = [1, 2]
list2 = [3, 4]
list1.extend(list2)
print(list1)
```

Options:

- [1, 2, [3,4]]
 - [1, 2, 3, 4]
 - [3, 4, 1, 2]
 - [1, 2, 3, 4, 5]
-

❓ Q18

What is the output?

```
numbers = [1, 2, 3, 4]
numbers.reverse()
print(numbers)
```

Options:

- [3, 2, 1, 4]
 - [4, 3, 2, 1]
 - [1, 2, 3, 4]
 - [2, 1, 3, 4]
-

❓ Q19

What is the output?

```
numbers = [3, 1, 4, 2]
numbers.sort()
print(numbers)
```

Options:

- [1, 2, 3, 4]
 - [4, 3, 2, 1]
 - [3, 1, 4, 2]
 - [1, 3, 2, 4]
-

❓ Q20

What is the output?

```
numbers = [1, 2, 3]
last = numbers.pop()
print(last)
print(numbers)
```

Options:

- 3 [1, 2]
 - 2 [1, 3]
 - 3 [1, 2, 3]
 - 1 [2, 3]
-

❓ Q21

What is the output?

```
original = [1, 2, 3]
copy_list = original.copy()
copy_list.append(4)
print(original)
print(copy_list)
```

Options:

- [1, 2, 3, 4] [1, 2, 3, 4]

- [1, 2, 3] [1, 2, 3, 4]
 - [1, 2, 3, 4] [1, 2, 3]
 - [1, 2, 3] [1, 2, 3]
-

❓ Q22

What is the output?

```
numbers = [3, 1, 4, 2]
numbers.reverse()
numbers.sort()
print(numbers)
```

Options:

- [4, 3, 2, 1]
 - [1, 2, 3, 4]
 - [2, 1, 3, 4]
 - [1, 3, 2, 4]
-

❓ Q23

What is the output?

```
numbers = [1, 2, 3]
numbers.clear()
print(numbers)
```

Options:

- [1, 2, 3, 4]
 - []
 - [0]
 - [1, 2]
-

❓ Q24

What is the output?

```
numbers = [1, 2, 2, 3, 2]
print(numbers.count(2))
```

Options:

- 1
 - 2
 - 3
 - 4
-

❓ Q25

What is the output?

```
fruits = ['apple', 'banana', 'cherry']
print(fruits.index('banana'))
```

Options:

- 0
 - 1
 - 2
 - ValueError
-

❓ Q26

What is the output?

```
numbers = [3, 1, 4, 2]
numbers.insert(1, 2)
print(numbers)
```

Options:

- [3, 2, 1, 4, 2]
 - [1, 3, 2, 4]
 - [1, 3, 4, 2]
 - [2, 3, 1, 4]
-

❓ Q27

What is the output?

```
numbers = [3, 1, 4, 2]
position = numbers.index(4)
print(position)
```

Options:

- 0
 - 1
 - 2
 - 3
-

❓ Q28

What will be the result?

```
my_tuple = (1, 2, 3)
my_tuple[1] = 4
```

Options:

- (1, 4, 3)
 - (1, 2, 3, 4)
 - TypeError
 - (1, 2, 3)
-

❓ Q29

```
my_tuple = (5, 10, 15)
print(my_tuple[1])
```

Options:

- 5
 - 10
 - 15
 - IndexError
-

❓ Q30

What is the output?

```
my_tuple = (10, 20, 30, 20)
print(my_tuple.index(20))
```

Options:

- 1
 - 2
 - 3
 - ValueError
-

❓ Q31

What is the output?

```
my_tuple = (1, 2, 2, 3)
print(my_tuple.count(2))
```

Options:

- 1
 - 2
 - 3
 - 0
-

❓ Q32

What is the output of slicing?

```
my_tuple = (10, 20, 30, 40, 50)
print(my_tuple[1:4])
```

Options:

- (20, 30, 40)
 - (10, 20, 30)
 - (30, 40, 50)
 - (20, 30, 40, 50)
-

❓ Q33

What is the output?

```
my_tuple = ('x', 'y', 'z')
print(len(my_tuple))
```

Options:

- 2
 - 3
 - 4
 - 0
-

❓ Q34

What is the output?

```
person = {'name': 'Alice', 'age': 25}  
print(person['age'])
```

Options:

- 'age'
 - 25
 - Alice
 - KeyError
-

❓ Q35

What is the output?

```
person = {'name': 'Bob', 'age': 30}  
print(person.get('city', 'Unknown'))
```

Options:

- Bob
 - 30
 - Unknown
 - None
-

❓ Q36

What is the output?

```
person = {'name': 'Charlie', 'age': 28}  
print(list(person.keys()))
```

Options:

- ['Charlie', 28]
 - ['name', 'age']
 - ['name': 'Charlie', 'age': 28]
 - ['name', 'age', 'Charlie', 28]
-

❓ Q37

What is the output?

```
person = {'name': 'Diana', 'age': 22}
for key, value in person.items():
    print(key, value)
```

Options:

- name Diana
age 22
 - ['name', 'age']
 - ('name', 'Diana')
('age', 22)
 - name: Diana
age: 22
-

❓ Q38

What is the output?

```
dict1 = {'a': 1, 'b': 2}
dict2 = {'b': 3, 'c': 4}
dict1.update(dict2)
print(dict1)
```

Options:

- {'a': 1, 'b': 2, 'c': 4}
 - {'a': 1, 'b': 3, 'c': 4}
 - {'a': 1, 'b': 2, 'b': 3, 'c': 4}
 - {'a': 1, 'c': 4}
-

❓ Q39

What is the output?

```
person = {'name': 'Eve', 'age': 28}
print(list(person.values()))
```

Options:

- ['name', 'age']
 - ['Eve', 28]
 - [{name: 'Eve'}, {age: 28}]
 - ['Eve: 28']
-

❓ Q40

What is the output?

```
person = {'name': 'Frank', 'age': 40}
keys = list(person.keys())
values = list(person.values())
combined = list(zip(keys, values))
print(combined)
```

Options:

- [('name', 'Frank'), ('age', 40)]
 - ['name', 'age', 'Frank', 40]
 - [('Frank', 'name'), (40, 'age')]
 - [('name', 'age'), ('Frank', 40)]
-

❓ Q41

What is the output?

```
person = {'name': 'Grace', 'age': 29}
age = person.pop('age')
print(age)
print(person)
```

Options:

- 29 {'name': 'Grace'}
 - 29 {'age': 29}
 - 'age' {'name': 'Grace', 'age': 29}
 - KeyError {'name': 'Grace'}
-

❓ Q42

What is the output?

```
dict1 = {'a': 1, 'b': 2}
dict2 = {'c': 3, 'd': 4}
combined = {**dict1, **dict2}
print(combined)
```

Options:

- {'a': 1, 'b': 2, 'c': 3, 'd': 4}
 - {'a': 1, 'b': 2}
 - {'c': 3, 'd': 4}
 - {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'a': 1}
-

❓ Q43

What is the output?

```
person = {'name': 'Henry', 'age': 45}
person.clear()
print(person)
```

Options:

- {'name': 'Henry'}
 - {}
 - {'age': 45}
 - {'name': 'Henry', 'age': 45}
-

❓ Q44

What is the output?

```
person = {'name': 'Ivy', 'age': 24}
keys = person.keys()
values = person.values()
print(list(keys), list(values))
```

Options:

- ['Ivy', 24]
- ['name', 'age'] ['Ivy', 24]

- ['name', 'age'] ['name', 'age']
 - ['Ivy'] [24]
-

❓ Q45

What is the output?

```
person = {'name': 'Jack', 'age': 30}
item = person.popitem()
print(item)
print(person)
```

Options:

- ('age', 30) {'name': 'Jack'}
 - ('name', 'Jack') {'age': 30}
 - ('name', 'Jack', 'age', 30) {}
 - ('age', 30) {'age': 30}
-

❓ Q46

What is the output?

```
name = "Mia"
age = 25
combined = {'name': name, 'age': age}
sentence = ', '.join([f'{k}: {v}' for k,v in combined.items()])
print(sentence)
```

Options:

- 'name: Mia, age: 25'
 - {'name': 'Mia', 'age': 25}
 - 'name Mia, age 25'
 - ['name: Mia', 'age: 25']
-

❓ Q47

What is the output?

```
my_set = {1, 2, 3}
my_set.add(4)
print(my_set)
```

Options:

- {1, 2, 3, 4}
 - [1, 2, 3, 4]
 - {1, 2, 3}
 - {4, 3, 2, 1}
-

❓ Q48

What will be the result?

```
my_frozenset = frozenset([1, 2, 3])
my_frozenset.add(4)
```

Options:

- frozenset({1,2,3,4})
 - frozenset({1,2,3})
 - AttributeError
 - TypeError
-

❓ Q49

What is the output?

```
my_set = {1, 2, 3}
my_set.remove(2)
print(my_set)
```

Options:

- {1, 3}
 - {1, 2, 3, 2}
 - {1, 2, 3, 4}
 - {2, 3}
-

❓ Q50

What is the output?

```
set1 = {1, 2, 3}
set2 = {2, 3, 4}
print(set1.intersection(set2))
```

Options:

- {1, 2, 3, 4}
 - {2, 3}
 - {1, 4}
 - {1, 2}
-

❓ Q51

What is the output?

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
print(set1.union(set2))
```

Options:

- {1, 2, 3, 4, 5}
 - {1, 2, 3}
 - {3}
 - {4,5}
-

❓ Q52

What is the output?

```
set1 = frozenset([1, 2, 3, 4])
set2 = frozenset([3, 4, 5])
print(set1.difference(set2))
```

Options:

- frozenset({1,2})
 - frozenset({5})
 - frozenset({3,4})
 - frozenset({1,2,5})
-

❓ Q53

What is the output?

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
print(set1.symmetric_difference(set2))
```

Options:

- {1,2,3,4,5}
 - {1,2,4,5}
 - {3}
 - {1,2,3,4}
-

❓ Q54

What is the output?

```
set1 = frozenset([1, 2])
set2 = frozenset([2, 3])
print(set1.union(set2))
```

Options:

- frozenset({1,2,3})
 - frozenset({2})
 - frozenset({1,3})
 - frozenset({1,2,2,3})
-

❓ Q55

What is the output?

```
text = "apple banana apple cherry"
unique_words = set(text.split())
print(unique_words)
```

Options:

- {'apple', 'banana', 'cherry'}
 - ['apple', 'banana', 'apple', 'cherry']
 - {'apple banana', 'cherry'}
 - {'apple banana apple cherry'}
-

❓ Q56

What is the output?

```
set1 = {1, 2}
set2 = {3, 4}
set1.update(set2)
print(set1)
```

Options:

- {1,2,3,4}
 - {1,2}
 - {3,4}
 - {1,2,3,4,5}
-

❓ Q57

What is the output?

```
set1 = frozenset([1, 2])
set2 = frozenset([1, 2, 3])
print(set1.issubset(set2))
```

Options:

- True
 - False
 - Error
 - None
-

❓ Q58

What is the difference between remove() and discard() in sets?

Options:

- remove() deletes all occurrences, discard() deletes one
 - remove() raises an error if element not found, discard() does not
 - remove() only removes integers, discard() any type
 - remove() adds an element, discard() removes an element
-

❓ Q59

What is the output?

```
original_set = {1, 2, 3}
copied_set = original_set.copy()
copied_set.add(4)
print(original_set)
print(copied_set)
```

Options:

- {1, 2, 3} {1, 2, 3, 4}
 - {1, 2, 3, 4} {1, 2, 3}
 - {1, 2, 3} {1, 2, 3}
 - {1, 2, 3,4} {1, 2, 3,4}
-

❓ Q60

What is the output?

```
person = {'name': 'Leo', 'age': 29}
keys = person.keys()
sentence = ', '.join(keys)
print(sentence)
```

Options:

- 'name, age'
 - 'Leo, 29'
 - ['name', 'age']
 - 'name: Leo, age: 29'
-

✓ The Answers

Q Nr.	Answer	Explanation
1	A	<code>upper()</code> converts all chars to uppercase. "hello world" -> "HELLO WORLD"
2	B	' '.join(['Hello', 'World']) -> "Hello World"
3	B	List comprehension <code>word.upper()</code> -> ["PYTHON", "IS", "FUN"]; joined by spaces -> "PYTHON IS FUN"
4	A	<code>replace("a", "o")</code> replaces all 'a's in "banana" -> "bonono"
5	B	' '.join(['P', 'y', 't', 'h', 'o', 'n']) -> "Python"
6	B	<code>split()</code> splits "Python is awesome" into ["Python", "is", "awesome"]
7	A	"{} is {} years old".format("Lily", 22) -> "Lily is 22 years old"
8	A	"OpenAI ChatGPT".startswith("Open") -> True
9	A	"Science" starts at index 5 in "Data Science"
10	B	"HELLO WORLD".lower() -> "hello world"
11	D	<code>replace("a", "o", 2)</code> changes first two 'a's -> "bonana"
12	A	"OpenAI ChatGPT".endswith("ChatGPT") -> True
13	A	<code>strip()</code> removes leading/trailing spaces. "Hello World " -> "Hello World"
14	A	<code>fruits.append('cherry')</code> -> ['apple', 'banana', 'cherry']
15	A	<code>remove(2)</code> removes first 2 -> [1, 3, 2]
16	A	<code>pop(2)</code> removes element at index 2 (3). Remaining [1, 2, 4]
17	B	<code>extend([3, 4])</code> -> [1, 2, 3, 4]
18	B	<code>reverse()</code> of [1, 2, 3, 4] -> [4, 3, 2, 1]
19	A	<code>sort()</code> of [3, 1, 4, 2] -> [1, 2, 3, 4]
20	A	<code>pop()</code> removes last element (3). Remaining [1, 2]
21	B	<code>copy()</code> creates separate list; original unmodified -> [1, 2, 3] and [1, 2, 3, 4]
22	B	After reverse [2, 4, 1, 3], then sort -> [1, 2, 3, 4]
23	B	<code>clear()</code> -> []
24	C	<code>count(2)</code> in [1, 2, 2, 3, 2] -> 3 times
25	B	'banana' is at index 1
26	A	Insert(1, 2) into [3, 1, 4, 2] -> [3, 2, 1, 4, 2]
27	C	<code>index(4)</code> in [3, 1, 4, 2] -> 2

Q Nr.	Answer	Explanation
28	C	Tuples are immutable -> `TypeError`
29	B	my_tuple=(5,10,15), my_tuple[1]=10`
30	A	first occurrence of 20 in (10,20,30,20) is index 1`
31	B	count(2) in (1,2,2,3) is 2`
32	A	my_tuple[1:4]=(20,30,40)`
33	B	len(('x','y','z'))=3`
34	B	person['age']=25`
35	C	person.get('city','Unknown') -> 'Unknown'`
36	B	keys() of {'name':'Charlie','age':28} -> ['name','age']`
37	A	items() print key value pairs: name Diana; age 22`
38	B	update merges dict2, 'b':2 replaced with 'b':3`
39	B	values() -> ['Eve',28]`
40	A	zip(keys,values)->[('name','Frank'),('age',40)]`
41	A	pop('age')=29, dict now {'name':'Grace'}`
42	A	{**dict1,**dict2} merges to {'a':1,'b':2,'c':3,'d':4}`
43	B	clear() -> {}`
44	B	keys=['name','age'], values=['Ivy',24]`
45	A	popitem() removes last inserted ('age',30)`
46	A	join formatted items -> 'name: Mia, age: 25'`
47	A	add(4) -> {1,2,3,4}`
48	C	frozenset has no add() -> `AttributeError`
49	A	remove(2) from {1,2,3} -> {1,3}`
50	B	intersection({1,2,3},{2,3,4}) = {2,3}`
51	A	union({1,2,3},{3,4,5})={1,2,3,4,5}`
52	A	difference([1,2,3,4],[3,4,5])= {1,2}`
53	B	symmetric_difference({1,2,3},{3,4,5})={1,2,4,5}`
54	A	union of frozenset([1,2]) and ([2,3])= frozenset({1,2,3})`
55	A	unique words from 'apple banana apple cherry' -> {'apple','banana','cherry'}`
56	A	update adds all elements -> {1,2,3,4}`

Q Nr.	Answer	Explanation
57	A	{1,2}.issubset({1,2,3}) = True`
58	B	remove() error if missing, discard() no error`
59	A	copy() separate set, original {1,2,3}, copy {1,2,3,4}`
60	A	join keys {'name','age'} -> 'name, age`

Licensed for: WBS Training AG
PYDASC71e-13

1.4 Input and Type Casting

Required Pre-Knowledge

- Understanding the `None` type in Python and how it behaves.
- Familiarity with boolean evaluation of empty data structures (lists, strings, dictionaries, sets).
- Awareness of common Python errors and exceptions: `SyntaxError`, `IndentationError`, `NameError`, `ValueError`.
- Understanding how `input()` works, how to parse user input, and how to handle conversion to int/float.
- Basic knowledge of Python functions and return values (including `None`).

Quiz

It contains 20 questions. Please answer the questions on separate paper or in an Excel file, and then compare your answers with those in the answers section after the quiz.

Q1

What is the output of the following code?

```
a = None  
print(a)
```

Options:

- `NoneType`
- `null`
- `None`
- `undefined`

Q2

What is the output of the following code?

```
a = None  
if a is None:  
    print("a is None")  
else:  
    print("a is not None")
```

Options:

- a is not None
 - a is None
 - True
 - False
-

❓ Q3

What is the output of the following code?

```
def greet():
    print("Hello!")

result = greet()
print(result)
```

Options:

- Hello! Hello!
 - Hello! None
 - Hello!
 - None
-

❓ Q4

What is the output of the following code?

```
def do_nothing():
    return None

result = do_nothing()
print(result is None)
```

Options:

- True
 - False
 - None
 - It raises an error
-

❓ Q5

What is the output?

```
my_list = []
```

```
if not my_list:  
    print("List is empty")  
else:  
    print("List is not empty")
```

Options:

- List is not empty
 - List is empty
 - No output
 - False
-

❓ Q6

What is the output?

```
text = ""  
if not text:  
    print("String is empty")  
else:  
    print("String is not empty")
```

Options:

- String is not empty
 - String is empty
 - No output
 - False
-

❓ Q7

What is the output?

```
my_dict = {}  
if not my_dict:  
    print("Dictionary is empty")  
else:  
    print("Dictionary is not empty")
```

Options:

- Dictionary is not empty
- Dictionary is empty
- No output
- False

❓ Q8

What is the output?

```
my_set = set()
if not my_set:
    print("Set is empty")
else:
    print("Set is not empty")
```

Options:

- Set is not empty
- Set is empty
- No output
- False

❓ Q9

What happens?

```
x = 10
if x > 5
    print("x is greater than 5")
```

Options:

- It prints x is greater than 5
- It prints nothing
- It raises a SyntaxError
- It raises a NameError

❓ Q10

What happens?

```
def greet():
    print("Hello")
    print("Welcome")
```

Options:

- It prints both "Hello" and "Welcome"

- It prints only "Hello"
 - It raises an **IndentationError**
 - It raises a **SyntaxError**
-

❓ Q11

What happens?

```
print(message)
```

Options:

- It prints **message**
 - It prints nothing
 - It raises a **NameError**
 - It raises a **SyntaxError**
-

❓ Q12

What happens?

```
message = "Hello, World!"  
print(message)
```

Options:

- **Hello, World!**
 - **Hello, World!'**
 - It raises a **SyntaxError**
 - It raises a **NameError**
-

❓ Q13

If user enters **Python**:

```
language = input("Enter a programming language: ")  
print("You entered:", language)
```

Options:

- Enter a programming language: Python You entered: Python
- You entered: Python

- Enter a programming language:
 - Python
-

❓ Q14

If user enters **Data**:

```
data = input()
print("Data entered:", data)
```

Options:

- Data entered: Data
 - Data
 - It prints nothing
 - Data entered:
-

❓ Q15

If user enters **John 25**:

```
name, age = input("Enter name and age: ").split()
print("Name:", name)
print("Age:", age)
```

Options:

- Name: John
Age: 25
 - Name: John 25
Age:
 - Name:
Age: John 25
 - It raises an error
-

❓ Q16

If user enters **3.14**:

```
pi = float(input("Enter the value of pi: "))
print("Pi is approximately:", pi)
```

Options:

- Pi is approximately: 3.14
 - Pi is approximately: "3.14"
 - Pi is approximately: 3
 - It raises an error
-

❓ Q17

If user enters abc:

```
number = int(input("Enter a number: "))
print("Number is:", number)
```

Options:

- It prints Number is: abc
 - It prints Number is: 100
 - It raises a ValueError
 - It prints Number is: 0
-

❓ Q18

If user enters Alice 30 Developer:

```
name, age = input("Enter name and age: ").split()
print(f"{name} is {age} years old")
```

Options:

- Alice is 30 years old
 - Alice 30 is Developer years old
 - It raises a ValueError
 - Alice is Developer years old
-

❓ Q19

If user presses Enter without typing:

```
age = int(input("Enter your age: "))
print("Next year you will be:", age + 1)
```

Options:

- Next year you will be: 1
 - Next year you will be:
 - It raises a ValueError
 - It raises an EOFError
-

❓ Q20

If user enters twenty:

```
age = int(input("Enter your age: "))
print("Next year you will be:", age + 1)
```

Options:

- Next year you will be: twenty1
 - Next year you will be: 21
 - It raises a ValueError
 - It prints Next year you will be: age + 1
-

✓ The Answers

Q Nr.	Answer	Explanation
1	C	Printing <code>None</code> results in <code>None</code>
2	B	<code>a is None</code> is True, so "a is None" printed
3	B	<code>greet()</code> prints "Hello!" and returns <code>None</code> , <code>print(result)</code> -> <code>None</code>
4	A	<code>do_nothing()</code> returns <code>None</code> , <code>is None</code> -> True
5	B	Empty list is Falsey, <code>if not my_list:</code> -> "List is empty"
6	B	Empty string is Falsey -> "String is empty"
7	B	Empty dict is Falsey -> "Dictionary is empty"
8	B	Empty set is Falsey -> "Set is empty"
9	C	Missing colon -> <code>SyntaxError</code>
10	C	Misaligned indentation -> <code>IndentationError</code>
11	C	<code>message</code> not defined -> <code>NameError</code>
12	C	Mismatched quotes -> <code>SyntaxError</code>
13	B	Input "Python" -> <code>print("You entered: Python")</code>
14	A	Input "Data" -> "Data entered: Data"
15	A	"John 25" splits into name="John", age="25"
16	A	Input "3.14" -> <code>float(3.14)</code> -> prints 3.14
17	C	<code>int("abc")</code> -> <code>ValueError</code>
18	C	3 inputs instead of 2 -> <code>ValueError</code>
19	C	<code>int("")</code> -> <code>ValueError</code>
20	C	<code>int("twenty")</code> -> <code>ValueError</code>



1.5 For-Loop

🔍 Required Pre-Knowledge

- Understanding of `for` loops in Python.
 - Familiarity with the `range()` function and its parameters: start, stop, step.
 - Ability to iterate over different data structures: strings, lists, tuples, dictionaries.
 - Knowledge of loop control statements: `break`, `continue`.
 - Understanding of for-else constructs.
 - Familiarity with functions like `enumerate()` and `zip()` in loops.
 - Understanding of list comprehensions and their equivalence to for loops.
 - Ability to identify syntax errors in loop syntax: missing colon, incorrect indentation.
 - Knowledge of nested loops and conditional statements within loops.
-

✓ Quiz

It contains 30 questions. Please answer the questions on separate paper or in an Excel file, and then compare your answers with those in the answers section after the quiz.

❓ Q1

What is the output of the following code?

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

Options:

- 1
 - 0 1
 - 0 1 2
 - 0 1 2 3
-

❓ Q2

What is the output of the following code?

```
for char in "Cat":
    print(char.upper())
```

Options:

- c a t
 - C A T
 - CAT
 - c A t
-

❓ Q3

What is the output of the following code?

```
for i in range(2, 5):  
    print(i)
```

Options:

- 1 2 3
 - 2 3 4
 - 2 3 4 5
 - 3 4 5
-

❓ Q4

What is the output of the following code?

```
for i in range(1, 3):  
    for j in range(1, 3):  
        print(i * j, end=' ')  
    print()
```

Options:

- 1 2
2 4
 - 1 1
2 2
 - 1 2
1 2
 - 2 4
2 4
-

❓ Q5

What is the output of the following code?

```
numbers = [1, 2, 3]
total = 0
for num in numbers:
    total += num
print(total)
```

Options:

- 3
 - 6
 - 1
 - 123
-

❓ Q6

What is the output of the following code?

```
pairs = [(1, 'a'), (2, 'b')]
for number, letter in pairs:
    print(number, letter)
```

Options:

- 1a
2b
 - (1, 'a')
(2, 'b')
 - 1 a
2 b
 - 1 a 2 b
-

❓ Q7

What is the output of the following code?

```
person = {'name': 'Alice', 'age': 25}
for key in person:
    print(key)
```

Options:

- Alice
25

- name
 - age
 - 'name'
 - 'age'
 - No output
-

❓ Q8

What is the output of the following code?

```
for i in range(3):
    print(i)
else:
    print("Done")
```

Options:

- 0
 - 1
 - 2
 - 0
 - 1
 - 2
 - Done
 - Done
 - No output
-

❓ Q9

What is the result of executing the following code?

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

Options:

- 0
 - 1
 - 2
 - SyntaxError
 - No output
 - RuntimeError
-

?

Q10

What is the output of the following code?

```
for i in range(3):
    if i == 1:
        print("One")
    else:
        print(i)
```

Options:

- 0
 - One
 - 2
 - 0
 - 1
 - 2
 - One
 - One
 - One
 - 1
 - 1
 - 1
-

?

Q11

What is the output of the following code?

```
colors = ['red', 'green']
shapes = ['circle', 'square']
for color in colors:
    for shape in shapes:
        print(color, shape)
```

Options:

- red circle green square
- red circle red square green circle green square
- red circle
- green square
- red circle
- green circle
- red square
- green square

❓ Q12

What is the output of the following code?

```
result = []
for x in range(3):
    result.append(x * 2)
print(result)
```

Options:

- [0, 1, 2]
- [0, 2, 4]
- [2, 4, 6]
- [0, 0, 0]

❓ Q13

What is the output of the following code?

```
for i in range(0, 10, 3):
    print(i, end=' ')
```

Options:

- 0 3 6 9
- 0 3 6
- 3 6 9
- 0 1 2 3 4 5 6 7 8 9

❓ Q14

What is the output of the following code?

```
data = [(1, 'a'), (2, 'b'), (3, 'c')]
for num, char in data:
    print(num, char)
```

Options:

- 1 a 2 b 3 c
- (1, 'a') (2, 'b') (3, 'c')
- 1 a

- 2 b
 - 3 c
 - a 1 b 2 c 3
-

❓ Q15

What is the output of the following code?

```
person = {'name': 'Bob', 'age': 30}
for value in person.values():
    print(value)
```

Options:

- name age
 - Bob 30
 - {'name': 'Bob', 'age': 30}
 - ['Bob', 30]
-

❓ Q16

What is the output of the following code?

```
for i in range(3):
    if i == 1:
        break
    print(i)
else:
    print("Completed")
```

Options:

- 0
 - 0 1
 - 0
 - 0
- Completed
-

❓ Q17

What is the result of executing the following code?

```
for i in range(2):  
    print(i)
```

Options:

- 0
 - 1
 - 0 1
 - 0
 - SyntaxError
-

❓ Q18

What is the output of the following code?

```
for i in range(3):  
    if i == 2:  
        print("Two")  
    else:  
        print(i)
```

Options:

- 0
 - 1
 - Two
 - 0
 - 1
 - 2
 - 0
 - Two
 - Two
 - Two
 - Two
-

❓ Q19

What is the output of the following code?

```
person = {'name': 'Charlie', 'age': 28}
for key, value in person.items():
    print(key, value)
```

Options:

- name Charlie age 28
 - {'name': 'Charlie', 'age': 28}
 - ['name', 'age']
 - Charlie 28
-

❓ Q20

What is the output of the following code?

```
fruits = ['apple', 'banana', 'cherry']
for index, fruit in enumerate(fruits):
    print(index, fruit)
```

Options:

- 0 apple
1 banana
2 cherry
 - 1 apple
2 banana
3 cherry
 - apple 0
banana 1
cherry 2
 - apple
banana
cherry
-

❓ Q21

What is the output of the following code?

```
for i in range(5, 0, -2):
    print(i, end=' ')
```

Options:

- 5 3 1
 - 5 4 3 2 1
 - 1 3 5
 - 5 3
-

❓ Q22

What is the output of the following code?

```
numbers = [1, 2, 3]
for i in numbers:
    numbers.append(i + 3)
    print(i)
    if len(numbers) > 5:
        break
print(numbers)
```

Options:

- 1 2 3
[1, 2, 3, 4, 5, 6]
 - 1 2 3 4 5
[1, 2, 3, 4, 5, 6]
 - 1 2 3 4
[1, 2, 3, 4, 5, 6]
 - 1 2 3 4 5
[1, 2, 3, 4, 5, 6, 7, 8]
-

❓ Q23

What is the output of the following code?

```
for i in range(2):
    print(i)
else:
    print("Loop completed")
```

Options:

- 0
1
- 0
Loop completed
- Loop completed

- No output
-

❓ Q24

What is the output of the following code?

```
grades = {'Alice': 85, 'Bob': 92}
for student, score in grades.items():
    print(student, score)
```

Options:

- Alice 85 Bob 92
 - {'Alice': 85} {'Bob': 92}
 - ['Alice', 'Bob']
 - Charlie 90
-

❓ Q25

What is the output of the following code?

```
names = ['Alice', 'Bob']
scores = [85, 92]
for name, score in zip(names, scores):
    print(f"{name} scored {score}")
```

Options:

- Alice scored 85
Bob scored 92
 - Alice Bob
85 92
 - Alice scored85 Bob scored 92
 - ['Alice scored 85', 'Bob scored 92']
-

❓ Q26

What is the output of the following code?

```
for i, letter in [(1, 'a'), (2, 'b'), (3, 'c')]:
    if i == 2:
        continue
    print(i, letter)
```

Options:

- 1 a
3 c
 - 1 a
2 b
3 c
 - 2 b
 - 1 a
2 b
-

❓ Q27

What is the output of the following code?

```
for i in range(2):
    print(i)
else:
    print("Finished")
```

Options:

- 0
1
 - 0
1
Finished
 - Finished
 - No output
-

❓ Q28

What is the output of the following code?

```
numbers = [1, 2, 3, 4]
for num in numbers:
    if num % 2 == 0:
        print(num)
```

Options:

- 1 3
- 2 4
- 1 2 3 4
- No output

❓ Q29

What is the output of the following code?

```
scores = {'Alice': 85, 'Bob': 92}
for student in scores:
    print(student)
```

Options:

- 85 92
- Alice Bob
- Alice: 85
 Bob: 92
- {'Alice': 85} {'Bob': 92}

❓ Q30

What is the output of the following code?

```
names = ['Alice', 'Bob']
scores_math = [90, 85]
scores_science = [95, 80]
for name, math, science in zip(names, scores_math, scores_science):
    print(f"{name}: Math={math}, Science={science}")
```

Options:

- Alice: Math=90, Science=95
 Bob: Math=85, Science=80
- Alice: Math=90
 Bob: Science=80
- Alice: Science=95
 Bob: Math=85
- Error

✓ The Answers

Question Nr.	Answer	Explanation
1	C	<code>range(3)</code> generates numbers from <code>0</code> up to, but not including, <code>3</code> . Therefore, the loop prints <code>0</code> , <code>1</code> , and <code>2</code> .
2	B	The <code>upper()</code> method converts each character to uppercase. The loop prints <code>C</code> , <code>A</code> , and <code>T</code> each on a new line.
3	B	<code>range(2, 5)</code> generates numbers starting at <code>2</code> up to, but not including, <code>5</code> . Thus, it prints <code>2</code> , <code>3</code> , and <code>4</code> .
4	A	The outer loop runs with <code>i = 1</code> and <code>i = 2</code> . The inner loop runs with <code>j = 1</code> and <code>j = 2</code> . The products are: $1*1=1$, $1*2=2$, $2*1=2$, $2*2=4$. Therefore, the output is <code>1 2</code> and <code>2 4</code> .
5	B	The loop adds each number in the list to <code>total: 1 + 2 + 3 = 6</code> .
6	C	The loop unpacks each tuple into <code>number</code> and <code>letter</code> , printing them separated by a space.
7	B	Iterating over a dictionary by default iterates over its keys. Therefore, it prints <code>name</code> and <code>age</code> .
8	B	After the loop completes without encountering a <code>break</code> , the <code>else</code> block is executed, printing <code>"Done"</code> .
9	B	A <code>for</code> loop requires a colon at the end of the <code>for</code> statement. Missing it results in a <code>SyntaxError</code> .
10	A	When <code>i = 0</code> : not equal to 1, prints <code>0</code> ; <code>i = 1</code> : equals 1, prints <code>"One"</code> ; <code>i = 2</code> : not equal to 1, prints <code>2</code> .
11	B	The nested loops generate all combinations: <code>red circle</code> , <code>red square</code> , <code>green circle</code> , <code>green square</code> .
12	B	The loop multiplies each <code>x</code> by <code>2</code> and appends it to <code>result</code> , resulting in <code>[0, 2, 4]</code> .
13	A	<code>range(0, 10, 3)</code> generates numbers starting at <code>0</code> , stepping by <code>3</code> , up to but not including <code>10</code> : <code>0, 3, 6, 9</code> .
14	C	The loop unpacks each tuple into <code>num</code> and <code>char</code> , printing them separated by a space on separate lines.
15	B	<code>person.values()</code> returns the values <code>'Bob'</code> and <code>30</code> . Therefore, it prints <code>Bob</code> and <code>30</code> .
16	C	The loop breaks when <code>i == 1</code> , so only <code>0</code> is printed. The <code>else</code> block is skipped because the loop was terminated by <code>break</code> .
17	D	The <code>print(i)</code> statement must be indented within the <code>for</code> loop. Lack of indentation results in a <code>SyntaxError</code> .

Question Nr.	Answer	Explanation
18	A	When <code>i = 0</code> : prints <code>0</code> ; <code>i = 1</code> : prints <code>1</code> ; <code>i = 2</code> : prints "Two".
19	A	The loop iterates over key-value pairs, printing <code>name Charlie</code> and <code>age 28</code> on separate lines.
20	A	<code>enumerate()</code> provides both the index and the item. The loop prints <code>0 apple</code> , <code>1 banana</code> , and <code>2 cherry</code> .
21	A	<code>range(5, 0, -2)</code> generates numbers starting at <code>5</code> , decreasing by <code>2</code> each step, until greater than <code>0</code> : <code>5, 3, 1</code> .
22	A	The loop breaks when <code>len(numbers) > 5</code> . Only <code>1, 2</code> , and <code>3</code> are printed before breaking. The final list is <code>[1, 2, 3, 4, 5, 6]</code> .
23	B	The loop runs for <code>i = 0</code> and <code>i = 1</code> , printing <code>0, 1</code> . Since the loop wasn't broken, the <code>else</code> block executes, printing "Loop completed".
24	A	The loop iterates over key-value pairs, printing <code>Alice 85</code> and <code>Bob 92</code> on separate lines.
25	A	<code>zip(names, scores)</code> pairs each name with its corresponding score, printing "Alice scored 85" and "Bob scored 92".
26	A	When <code>i = 1</code> : skips printing; <code>i = 2</code> : skips printing; <code>i = 3</code> : prints <code>3 c</code> . However, according to the code, it should print <code>1 a</code> and <code>3 c</code> . The correct answer is A.
27	B	After completing the loop without a <code>break</code> , the <code>else</code> block executes, printing "Finished".
28	B	The loop prints numbers that are even (<code>2</code> and <code>4</code>).
29	B	Iterating over a dictionary without <code>.values()</code> or <code>.items()</code> iterates over its keys, printing <code>Alice</code> and <code>Bob</code> .
30	A	<code>zip(names, scores_math, scores_science)</code> pairs each name with corresponding math and science scores, printing "Alice: Math=90, Science=95" and "Bob: Math=85, Science=80".

1.6 While-Loop

Required Pre-Knowledge

- **Python While Loops:** Understanding the syntax and basic structure of while loops.
- **List Operations:** Familiarity with methods like `append()`, `pop()`, `remove()`, and `insert()`.
- **Tuple Operations:** Understanding that tuples are immutable and how to access their elements.
- **Dictionary Operations:** Knowledge of accessing keys, using methods like `keys()`, `values()`, `get()`, `pop()`, and `update()`.
- **Control Flow Statements:** Understanding `break` and `continue` within loops.
- **Loop Else Clause:** Knowing how the `else` block works with loops.
- **Nested Loops:** Ability to work with loops within loops.
- **Error Identification:** Recognizing common syntax errors in loop structures.
- **List Comprehensions:** Understanding equivalent for loops for list comprehensions.
- **List Slicing and Indexing:** Proficiency in accessing and modifying list elements using indices.
- **Mutable vs Immutable Structures:** Knowing the difference between lists (mutable) and tuples (immutable).
- **Loop Termination Conditions:** Understanding how and when loops terminate.
- **Handling Multiple Lists:** Iterating over multiple lists simultaneously.
- **Loop Flags:** Using flags to control loop execution.
- **Counting Elements:** Using methods like `count()` to tally occurrences within lists.
- **List Reversing and Sorting:** Using `reverse()` and `sort()` methods effectively.

Quiz

It contains 20 questions. Please answer the questions on separate paper or in an Excel file, and then compare your answers with those in the answers section after the quiz.

Q1

What is the output of the following code?

```
count = 0
while count < 3:
    print(count)
    count += 1
```

Options:

- 1 2 3
- 0 1 2 3
- 0 1 2
- 0 1

❓ Q2

What is the output of the following code?

```
fruits = ['apple', 'banana', 'cherry']
while fruits:
    print(fruits.pop())
```

Options:

- apple banana cherry
- cherry banana apple
- apple banana
- cherry apple

❓ Q3

What is the output of the following code?

```
i = 1
while i <= 2:
    j = 1
    while j <= 2:
        print(i * j, end=' ')
        j += 1
    print()
    i += 1
```

Options:

- 1 2
2 4
- 1 1
2 2
- 1 2
3 4
- 2 4
2 4

❓ Q4

What is the output of the following code?

```
data = (1, 2, 3)
index = 0
while index < len(data):
    print(data[index])
    index += 1
```

Options:

- 1 2 3 4
 - 1 2 3
 - 0 1 2
 - 1 3 2
-

❓ Q5

What is the output of the following code?

```
person = {'name': 'Alice', 'age': 30}
keys = list(person.keys())
index = 0
while index < len(keys):
    print(keys[index])
    index += 1
```

Options:

- Alice 30
 - name age
 - ['name', 'age']
 - name
 age
-

❓ Q6

What is the output of the following code?

```
count = 0
while count < 3:
    print(count)
    count += 1
else:
    print("Finished")
```

Options:

- 0 1 2
 Finished
 - 0 1 2
 - Finished
 - No output
-

❓ Q7

What is the result of executing the following code?

```
count = 0
while count < 3
    print(count)
    count += 1
```

Options:

- 0 1 2
 - 0 1
 - 0
 - SyntaxError
-

❓ Q8

What is the output of the following code?

```
count = 0
while count < 5:
    if count == 3:
        print("Three")
    else:
        print(count)
    count += 1
```

Options:

- 0 1 2 Three 4
 - 0 1 2 3 4
 - 0 1 2 Three Three
 - Three 1 2 3 4
-

❓ Q9

What is the output of the following code?

```
numbers = [1, 2, 3, 4, 5]
i = 0
while i < len(numbers):
    if numbers[i] % 2 == 0:
        print(numbers[i])
    i += 1
```

Options:

- 1 3 5
 - 2 4
 - 1 2 3 4 5
 - 0 2 4
-

❓ Q10

Which of the following while loops is equivalent to the given for loop?

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

Options:

- []

```
i = 0
while i <= 3:
    print(i)
    i += 1
```

- []

```
i = 1
while i < 3:
    print(i)
    i += 1
```

- []

```
i = 0
while i < 3:
    print(i)
```

```
i += 1
```

- []

```
i = 0
while i < 4:
    print(i)
    i += 1
```

❓ Q11

What is the output of the following code?

```
count = 0
while count < 3:
    print(count)
    count += 1
    if count == 2:
        break
else:
    print("Completed")
```

Options:

- 0 1 2
- 0 1
- 0 1 Completed
- 0 1

❓ Q12

What is the output of the following code?

```
person = {'name': 'Bob', 'age': 25}
keys = list(person.keys())
i = 0
while i < len(keys):
    key = keys[i]
    print(key, person[key])
    i += 1
```

Options:

- name Bob age 25
 - Bob 25
 - name Bob
age 25
 - {'name': 'Bob'}
{'age': 25}
-

❓ Q13

Which for loop is equivalent to the following while loop?

```
i = 0
while i < 3:
    print(i)
    i += 1
```

Options:

- []

```
for i in range(1, 4):
    print(i)
```

- []

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

- []

```
for i in range(0, 3, 2):
    print(i)
```

- []

```
for i in range(0, 4):
    print(i)
```

❓ Q14

What is the output of the following code?

```
data = (10, 20, 30)
i = 0
while i < len(data):
    print(data[i])
    i += 1
```

Options:

- 10 20 30 40
 - 10 20 30
 - 10 30 20
 - 10 20
-

❓ Q15

What is the output of the following code?

```
numbers = [1, 2, 3, 4]
i = 0
while i < len(numbers):
    if numbers[i] % 2 == 0:
        i += 1
        continue
    print(numbers[i])
    i += 1
```

Options:

- 1 3
 - 2 4
 - 1 2 3 4
 - 1 3 5
-

❓ Q16

What is the output of the following code?

```
i = 0
while i < 2:
    print(i)
    i += 1
else:
    print("Loop completed")
```

Options:

- 0 1
 - 0 1 Loop completed
 - Loop completed
 - 0 1 2
-

❓ Q17

What is the result of executing the following code?

```
count = 0
while count < 3 print(count)
    count += 1
```

Options:

- 0 1 2
 - 0 1
 - 0 1 2 3
 - SyntaxError
-

❓ Q18

What is the output of the following code?

```
i = 0
while i < 5:
    if i == 3:
        print("Reached three")
        break
    print(i)
    i += 1
else:
    print("Completed")
```

Options:

- 0 1 2 Reached three Completed
 - 0 1 2 Reached three
 - 0 1 2 3
 - 0 1 2 3 Completed
-

❓ Q19

What is the output of the following code?

```
person = {'name': 'Diana', 'age': 22}
keys = list(person.keys())
i = 0
while i < len(keys):
    key = keys[i]
    print(key, person[key])
    if key == 'age':
        person['age'] = 23
    i += 1
print(person)
```

Options:

- name Diana age 22
{'name': 'Diana', 'age': 23}
 - name Diana age 22
{'name': 'Diana', 'age': 22}
 - name Diana
age 22
{'name': 'Diana', 'age': 23}
 - name Diana
age 22
{'name': 'Diana', 'age': 22}
-

❓ Q20

What is the output of the following code?

```
numbers = [2, 4, 6, 8]
count = 0
while count < len(numbers):
    if numbers[count] % 2 == 0:
        print(numbers[count])
    count += 1
```

Options:

- 2 4
 - 2 4 6 8
 - 4 6 8
 - No output
-

Licensed for: WBS Training AG
PYDASC71e-13

✓ The Answers

Question Nr.	Answer	Explanation
1	C	The loop starts with <code>count = 0</code> and runs while <code>count < 3</code> . It prints <code>0</code> , increments to <code>1</code> , prints <code>1</code> , increments to <code>2</code> , prints <code>2</code> , and then increments to <code>3</code> . Since <code>3</code> is not less than <code>3</code> , the loop exits. Hence, the output is <code>0 1 2</code> .
2	B	The <code>pop()</code> method removes and returns the last item from the list. The loop continues until the list is empty. It prints <code>cherry</code> , then <code>banana</code> , and finally <code>apple</code> .
3	A	For <code>i = 1</code> , <code>j</code> iterates from <code>1</code> to <code>2</code> , printing <code>1*1=1</code> and <code>1*2=2</code> . For <code>i = 2</code> , <code>j</code> iterates from <code>1</code> to <code>2</code> , printing <code>2*1=2</code> and <code>2*2=4</code> . Therefore, the output is <code>1 2</code> and <code>2 4</code> .
4	B	The tuple <code>data</code> contains <code>(1, 2, 3)</code> . The loop iterates over each index from <code>0</code> to <code>2</code> , printing <code>1</code> , <code>2</code> , and <code>3</code> sequentially.
5	D	<code>person.keys()</code> returns <code>['name', 'age']</code> . The loop iterates over the list of keys, printing <code>name</code> and then <code>age</code> on separate lines.
6	A	The <code>else</code> block executes after the loop completes normally (without a <code>break</code>). It prints <code>0, 1, 2</code> , and then "Finished".
7	D	A <code>while</code> loop requires a colon at the end of the condition. Missing it results in a <code>SyntaxError</code> .
8	A	The loop prints numbers from <code>0</code> to <code>4</code> . When <code>count</code> is <code>3</code> , it prints "Three" instead of <code>3</code> .
9	B	The loop checks each number for evenness. It prints <code>2</code> and <code>4</code> as they are the even numbers in the list.
10	C	The for loop iterates <code>i</code> from <code>0</code> to <code>2</code> . The equivalent while loop starts with <code>i = 0</code> and runs while <code>i < 3</code> , printing <code>i</code> and incrementing it by <code>1</code> each time.
11	D	The loop prints <code>0</code> and <code>1</code> . When <code>count</code> becomes <code>2</code> , the <code>break</code> statement exits the loop. The <code>else</code> block does not execute because the loop was terminated by <code>break</code> .
12	C	The loop iterates over the keys <code>'name'</code> and <code>'age'</code> , printing each key with its corresponding value on separate lines.
13	B	The while loop iterates <code>i</code> from <code>0</code> to <code>2</code> . The equivalent for loop is <code>for i in range(3)</code> , iterating <code>i</code> as <code>0, 1, and 2</code> .
14	B	The loop iterates over each element in the tuple <code>data</code> , printing <code>10, 20, and 30</code> .
15	A	The loop prints odd numbers by skipping even ones. It prints <code>1</code> and <code>3</code> .
16	B	After printing <code>0</code> and <code>1</code> , the loop condition <code>i < 2</code> fails, and the <code>else</code> block executes, printing "Loop completed".

Question Nr.	Answer	Explanation
17	D	The <code>while</code> statement is missing a colon after the condition, resulting in a <code>SyntaxError</code> .
18	B	The loop prints <code>0</code> , <code>1</code> , and <code>2</code> . When <code>i == 3</code> , it prints "Reached three" and breaks the loop. The <code>else</code> block does not execute.
19	A	The loop prints <code>name Diana</code> and <code>age 22</code> . When the key is ' <code>age</code> ', it updates <code>person['age']</code> to <code>23</code> . After the loop, the updated dictionary is <code>{'name': 'Diana', 'age': 23}</code> .
20	B	All numbers in the list are even. The loop prints <code>2</code> , <code>4</code> , <code>6</code> , and <code>8</code> .

Licensed for: WBS Training AG
PYDASC71e-13

2.1 Comprehensions

Required Pre-Knowledge

- Understanding of Python list, dict, and set comprehension syntax
 - Familiarity with basic Python control structures (if, else)
 - Knowledge of basic Python functions (`range`, `print`, `zip`, `count`, custom functions)
 - Understanding of dictionary merging, conditional filtering, and nested comprehensions
 - Awareness of difference between lists, sets, and dictionaries, and how comprehensions remove duplicates (in sets)
 - Grasping how conditions, loops, and multiple iterables work in comprehensions
-

Quiz

It contains 29 questions. Please answer the questions on separate paper or in an Excel file, and then compare your answers with those in the answers section after the quiz.

Q1

What is the output of the following code?

```
squares = [x**2 for x in range(3)]  
print(squares)
```

Options:

- [0, 1, 4, 9]
- [1, 4, 9]
- [0, 1, 4]
- [1, 2, 3]

Q2

What is the output of the following code?

```
even_squares = [x**2 for x in range(5) if x % 2 == 0]  
print(even_squares)
```

Options:

- [0, 1, 4, 9, 16]
- [0, 4, 16]
- [0, 1, 4]

- [1, 3, 5]

❓ Q3

What is the output of the following code?

```
matrix = [[x for x in range(2)] for y in range(3)]
print(matrix)
```

Options:

- [[0, 1], [0, 1], [0, 1]]
- [[0, 1, 2], [0, 1, 2], [0, 1, 2]]
- [[0], [1], [2]]
- [[0, 1], [0, 1, 2], [0, 1]]

❓ Q4

What is the output of the following code?

```
result = [x for x in range(10) if x > 5 if x % 2 == 0]
print(result)
```

Options:

- [6, 8, 10]
- [6, 8]
- [6, 7, 8, 9]
- [6, 8, 10, 12]

❓ Q5

What is the output of the following code?

```
matrix = [[1, 2], [3, 4]]
flattened = [num for row in matrix for num in row]
print(flattened)
```

Options:

- [[1, 2], [3, 4]]
- [1, 2, 3, 4]
- [[1, 2, 3, 4]]
- [1, 2, [3, 4]]

?

Q6

What is the output of the following code?

```
pairs = [(x, y) for x in range(2) for y in range(2)]  
print(pairs)
```

Options:

- [(0, 0), (1, 1)]
- [(0, 0), (0, 1), (1, 0), (1, 1)]
- [(0, 0), (1, 1), (0, 1), (1, 0)]
- [(0, 0), (0, 1), (1, 0)]

?

Q7

What is the output of the following code?

```
result = ["even" if x % 2 == 0 else "odd" for x in range(3)]  
print(result)
```

Options:

- ["even", "even", "even"]
- ["odd", "even", "odd"]
- ["odd", "odd", "odd"]
- ["even", "odd", "even"]

?

Q8

What is the output of the following code?

```
list1 = [1, 2]  
list2 = [3, 4]  
combined = [x + y for x in list1 for y in list2]  
print(combined)
```

Options:

- [4, 6]
- [4, 5, 5, 6]
- [1, 2, 3, 4]
- [3, 4, 5, 6]

?

Q9

What is the output of the following code?

```
result = [x for x in range(10) if x > 5 and x % 2 == 0]
print(result)
```

Options:

- [6, 8, 10]
- [6, 8]
- [6, 7, 8, 9]
- [6, 8, 10, 12]

❓ Q10

What is the output of the following code?

```
def cube(x):
    return x**3

cubes = [cube(x) for x in range(3)]
print(cubes)
```

Options:

- [1, 8, 27]
- [0, 1, 8]
- [0, 1, 27]
- [1, 8, 64]

❓ Q11

What is the output of the following code?

```
original = {'a': 1, 'b': 2, 'c': 3}
swapped = {value: key for key, value in original.items()}
print(swapped)
```

Options:

- {'a': 1, 'b': 2, 'c': 3}
- {1: 'a', 2: 'b', 3: 'c'}
- {'1': 'a', '2': 'b', '3': 'c'}
- ['a', 'b', 'c']

❓ Q12

What is the output of the following code?

```
original = {'apple': 2, 'banana': 3, 'cherry': 1}  
filtered = {k: v for k, v in original.items() if v > 1}  
print(filtered)
```

Options:

- {'apple': 2, 'banana': 3}
- {'banana': 3, 'cherry': 1}
- {'apple': 2, 'banana': 3, 'cherry': 1}
- {'cherry': 1}

❓ Q13

What is the output of the following code?

```
original = {'a': 1, 'b': 2, 'c': 3}  
new_dict = {k: ('even' if v % 2 == 0 else 'odd') for k, v in original.items()}  
print(new_dict)
```

Options:

- {'a': 'odd', 'b': 'even', 'c': 'odd'}
- {'a': 1, 'b': 2, 'c': 3}
- {'a': 'even', 'b': 'odd', 'c': 'even'}
- {'a': 'odd', 'b': 'odd', 'c': 'odd'}

❓ Q14

What is the output of the following code?

```
dict1 = {'a': 1, 'b': 2}  
dict2 = {'c': 3, 'd': 4}  
merged = {**dict1, **dict2}  
print(merged)
```

Options:

- {'a': 1, 'b': 2}
- {'c': 3, 'd': 4}
- {'a': 1, 'b': 2, 'c': 3, 'd': 4}
- {'a': 1, 'b': 2, 'c': 3}

❓ Q15

What is the output of the following code?

```
original = {'x': 10, 'y': 20, 'z': 30}
inverted = {v: k for k, v in original.items() if v > 15}
print(inverted)
```

Options:

- {10: 'x', 20: 'y', 30: 'z'}
- {20: 'y', 30: 'z'}
- {10: 'x', 30: 'z'}
- {20: 'y'}

❓ Q16

What is the output of the following code?

```
keys = ['a', 'b', 'c']
values = [1, 2, 3]
d = {k: v**2 for k, v in zip(keys, values)}
print(d)
```

Options:

- {'a': 1, 'b': 2, 'c': 3}
- {'a': 1, 'b': 4, 'c': 9}
- {'a': 2, 'b': 4, 'c': 6}
- {'a': '1', 'b': '4', 'c': '9'}

❓ Q17

What is the output of the following code?

```
grades = {'Alice': 85, 'Bob': 67, 'Charlie': 92}
status = {k: ('Pass' if v >= 70 else 'Fail') for k, v in grades.items()}
print(status)
```

Options:

- {'Alice': 'Pass', 'Bob': 'Pass', 'Charlie': 'Pass'}
- {'Alice': 'Pass', 'Bob': 'Fail', 'Charlie': 'Pass'}
- {'Alice': 'Fail', 'Bob': 'Fail', 'Charlie': 'Fail'}
- {'Alice': 85, 'Bob': 67, 'Charlie': 92}

❓ Q18

What is the output of the following code?

```
keys = ['a', 'b']
values = [1, 2]
nested = {k: {v: v*10 for v in values} for k in keys}
print(nested)
```

Options:

- {'a': {1: 10, 2: 20}, 'b': {1: 10, 2: 20}}
- {'a': [10, 20], 'b': [10, 20]}
- {'a': {1: '10', 2: '20'}, 'b': {1: '10', 2: '20'}}
- {'a': {1: 10, 2: 20}, 'b': {1: 10, 2: 20}, 'c': {1: 10, 2: 20}}

?

Q19

What is the output of the following code?

```
scores = {'Alice': 85, 'Bob': 67, 'Charlie': 92, 'David': 45}
passed = {k: v for k, v in scores.items() if v >= 70}
print(passed)
```

Options:

- {'Alice': 85, 'Bob': 67}
- {'Alice': 85, 'Charlie': 92}
- {'Charlie': 92, 'David': 45}
- {'Alice': 85, 'Bob': 67, 'Charlie': 92}

?

Q20

What is the output of the following code?

```
words = ['apple', 'banana', 'apple', 'cherry', 'banana', 'apple']
frequency = {word: words.count(word) for word in words}
print(frequency)
```

Options:

- {'apple': 3, 'banana': 2, 'cherry': 1}
- {'apple': 1, 'banana': 1, 'cherry': 1}
- {'apple': 3, 'banana': 2}
- {'cherry': 1}

?

Q21

What is the output of the following code?

```
nums = [1, 2, 2, 3, 4, 4, 5]
unique = {x for x in nums}
print(unique)
```

Options:

- [1, 2, 3, 4, 5]
- {1, 2, 3, 4, 5}
- {'1', '2', '3', '4', '5'}
- [1, 2, 2, 3, 4, 4, 5]

❓ Q22

What is the output of the following code?

```
nums = [1, 2, 3, 4]
squares = {x**2 for x in nums if x % 2 == 0}
print(squares)
```

Options:

- {1, 4, 9, 16}
- {16, 4}
- {1, 9}
- [1, 4, 16]

❓ Q23

What is the output of the following code?

```
def square(x):
    return x * x

nums = [1, 2, 3]
squared_set = {square(x) for x in nums}
print(squared_set)
```

Options:

- [1, 4, 9]
- {1, 4, 9}
- {'1', '4', '9'}
- {1, 4, 9, 16}

❓ Q24

What is the output of the following code?

```
nums = range(5)
s = {x for x in nums if x != 3}
print(s)
```

Options:

- {0, 1, 2, 3, 4}
- {0, 1, 2, 4}
- {1, 2, 4}
- {0, 2, 4}

❓ Q25

What is the output of the following code?

```
word = "hello"
unique_chars = {char for char in word}
print(unique_chars)
```

Options:

- {'h', 'e', 'l', 'l', 'o'}
- {'h', 'e', 'l', 'o'}
- ['h', 'e', 'l', 'o']
- ['h', 'e', 'l', 'l', 'o']

❓ Q26

What is the output of the following code?

```
sentence = "Mississippi"
unique_letters = {char for char in sentence}
print(unique_letters)
```

Options:

- {'M', 'i', 's', 'p'}
- {'m', 'i', 's', 'p'}
- {'M', 'I', 'S', 'P'}
- {'M', 'I', 'S', 'P', 'i', 's', 'p'}

?

Q27

What is the output of the following code?

```
numbers = [1, 2, 3, 2, 1]
squares = {x**2 for x in numbers}
print(squares)
```

Options:

- [1, 4, 9, 4, 1]
- {1, 4, 9}
- {1, 4, 9, 16}
- {'1', '4', '9'}

?

Q28

What is the output of the following code?

```
result = {x if x % 2 == 0 else -x for x in range(3)}
print(result)
```

Options:

- {0, -1, 2}
- {0, 1, 2}
- {0, -1, -2}
- {-0, -1, 2}

?

Q29

What is the output of the following code?

```
sentence = "Hello World"
vowels = {'a', 'e', 'i', 'o', 'u'}
consonants = {char for char in sentence.lower() if char.isalpha() and char not
in vowels}
print(consonants)
```

Options:

- {'h', 'l', 'w', 'r', 'd'}
- {'H', 'L', 'W', 'R', 'D'}
- {'h', 'e', 'l', 'o', 'w', 'r', 'd'}
- {'h', 'l', 'w', 'r', 'd', 'o'}

Licensed for: WBS Training AG
PYDASC71e-13

✓ The Answers

Question Nr.	Answer	Explanation
1	C	Iterates 0,1,2, squares each → <code>[0,1,4]</code>
2	B	Even numbers in range(5):0,2,4 → <code>[0,4,16]</code>
3	A	3 rows, each inner comp <code>[0,1]</code>
4	B	Numbers>5 & even:6,8
5	B	Flatten matrix using nested for
6	B	All pairs from two ranges
7	D	For 0 even, 1 odd, 2 even
8	B	Combines each from list1 to each of list2
9	B	Numbers>5 and even:6,8
10	B	Cubes of 0,1,2 are 0,1,8
11	B	Swap keys & values
12	A	Values>1 only
13	A	Conditional even/odd values
14	C	Merging dicts
15	B	Values>15 swapped
16	B	Squared values
17	B	Conditionally "Pass" if \geq 70 else "Fail"
18	A	Nested dict comp
19	B	Filter by $v \geq 70$
20	A	Counting occurrences
21	B	Set removes duplicates
22	B	Squares of even only
23	B	squares of 1,2,3
24	B	Exclude x=3 from range(5)
25	B	Unique chars of "hello"
26	A	Unique chars of "Mississippi"
27	B	Squares of 1,2,3 remove duplicates
28	A	if-even→x else -x for x=0,1,2
29	A	Consonants from "hello world"



2.2 Exceptions

🔍 Required Pre-Knowledge

- Understanding Python `try` and `except` blocks for exception handling
- Knowing how division operations work in Python, and what happens when no error occurs vs when a `ZeroDivisionError` might occur
- Recognizing that if no exception is raised in the `try` block, the `except` block is not executed
- Understanding how file operations are performed using the `open()` function in Python
- Knowing what a `FileNotFoundException` exception is and how it is raised when trying to open a non-existent file
- Understanding that `raise` inside an `except` block can re-raise the caught exception

✓ Quiz

It contains 75 questions. Please answer the questions on separate paper or in an Excel file, and then compare your answers with those in the answers section after the quiz.

❓ Q1:

What is the output of the following code?

```
try:  
    print("Attempting division")  
    result = 10 / 2  
    print("Result is", result)  
except ZeroDivisionError:  
    print("Cannot divide by zero")
```

- Attempting division
- Attempting division
Result is 5.0
- Cannot divide by zero
- Attempting division
Cannot divide by zero

❓ Q2:

What is the output of the following code?

```
try:  
    print("Trying to open file")  
    f = open("non_existent_file.txt", "r")
```

```
except FileNotFoundError:  
    print("File not found, re-raising exception")  
    raise
```

- Trying to open file
File not found, re-raising exception
- Trying to open file
File not found, re-raising exception
It raises a FileNotFoundError
- File not found, re-raising exception
- It raises an error before printing anything

?

Q3:

What is the output of the following code?

```
try:  
    print("Start")  
    x = 5 / 0  
except ValueError:  
    print("ValueError caught")
```

- Start
ValueError caught
- Start
It raises a ZeroDivisionError
- It raises a ValueError
- ValueError caught

?

Q4:

What is the output of the following code?

```
try:  
    print("Attempting operation")  
    x = int("abc")  
except ValueError:  
    pass  
print("Operation completed")
```

- Attempting operation
Operation completed
- Operation completed
- It raises a ValueError
- Attempting operation

It raises a ValueError

Q5:

What is the output of the following code?

```
try:  
    print("Starting process")  
    raise Exception("Custom error")  
except Exception as e:  
    print("Caught an exception:", e)
```

- Starting process
Caught an exception: Custom error
- Caught an exception: Custom error
- Starting process
It raises an exception
- It raises an exception without printing anything

Q6:

What is the output of the following code?

```
try:  
    print("Attempting risky operation")  
    x = undefined_variable  
except:  
    print("An error occurred")
```

- Attempting risky operation
An error occurred
- It raises a NameError
- An error occurred
- Attempting risky operation
It raises a NameError

Q7:

What is the output of the following code?

```
def divide(a, b):  
    return a / b  
  
try:  
    print("Result:", divide(10, 0))  
except ZeroDivisionError:
```

```
print("Cannot divide by zero")
```

- Result: inf
- Result: 0
- Cannot divide by zero
- It raises a ZeroDivisionError

?

Q8:

What is the output of the following code?

```
try:  
    print("Attempting operation")  
    x = 5 + "5"  
except Exception:  
    print("An exception occurred")
```

- Attempting operation
An exception occurred
- It raises a TypeError
- Attempting operation
- An exception occurred

?

Q9:

What is the output of the following code?

```
try:  
    print("Accessing attribute")  
    obj = 5  
    print(obj.attribute)  
except AttributeError:  
    print("AttributeError caught")
```

- Accessing attribute
AttributeError caught
- Accessing attribute
It raises an AttributeError
- AttributeError caught
- It raises an error before printing

?

Q10:

What is the output of the following code?

```
try:  
    print("Attempting risky operation")  
    x = int("not a number")  
except ValueError:  
    pass  
print("Operation completed")
```

- Attempting risky operation
Operation completed
- Operation completed
- It raises a ValueError
- Attempting risky operation
It raises a ValueError

?

Q11:

What is the output of the following code?

```
try:  
    print("Running code")  
    x = 1 / "a"  
except:  
    print("An error occurred")
```

- Running code
An error occurred
- It raises a TypeError
- An error occurred
- Running code

?

Q12:

What is the output of the following code?

```
try:  
    print("Accessing dictionary key")  
    d = {'a': 1, 'b': 2}  
    print(d['c'])  
except KeyError:  
    print("KeyError caught")
```

- Accessing dictionary key
It raises a KeyError
- Accessing dictionary key
KeyError caught

- KeyError caught
- It raises an error before printing

?

Q13:

What is the output of the following code?

```
try:  
    print("Attempting risky operation")  
    x = 1 / 0  
except ZeroDivisionError:  
    pass  
print("Operation completed")
```

- Attempting risky operation
Operation completed
- Operation completed
- It raises a ZeroDivisionError
- Attempting risky operation
It raises a ZeroDivisionError

?

Q14:

What is the output of the following code?

```
try:  
    print("Starting computation")  
    result = 10 / 0  
except ValueError:  
    print("ValueError caught")  
except Exception:  
    print("General Exception caught")
```

- Starting computation
ValueError caught
- Starting computation
General Exception caught
- General Exception caught
- It raises a ZeroDivisionError

?

Q15:

What is the output of the following code? (when interrupted by Ctrl+C)

```
try:  
    while True:
```

```
    pass
except KeyboardInterrupt:
    print("KeyboardInterrupt caught")
```

- It prints nothing and exits
- It prints "KeyboardInterrupt caught"
- It raises a KeyboardInterrupt
- It prints an error message

❓ Q16:

What is the output of the following code?

```
try:
    print("Attempting conversion")
    x = float("NaN")
    print("Conversion successful")
except ValueError:
    print("ValueError caught")
```

- Attempting conversion
ValueError caught
- Attempting conversion
Conversion successful
- It raises a ValueError
- Conversion successful

❓ Q17:

What is the output of the following code?

```
try:
    print("Attempting risky computation")
    x = 10 / "2"
except TypeError:
    pass
print("Computation done")
```

- Attempting risky computation
Computation done
- Computation done
- It raises a TypeError
- Attempting risky computation
It raises a TypeError

❓ Q18:

What is the output of the following code?

```
try:  
    print("Accessing list element")  
    lst = [1, 2, 3]  
    print(lst[5])  
except IndexError:  
    print("Index out of range")  
except ZeroDivisionError:  
    print("Division by zero")
```

- Accessing list element
Index out of range
- Accessing list element
Division by zero
- Accessing list element
3
- It raises an IndexError

❓ Q19:

What is the output of the following code?

```
try:  
    print("Performing operation")  
    x = int("abc")  
except ValueError:  
    print("Value error occurred")  
except Exception:  
    print("General exception")
```

- Performing operation
Value error occurred
- Performing operation
General exception
- Value error occurred
- Performing operation
It raises a ValueError

❓ Q20:

What is the output of the following code?

```
try:  
    print("Attempting operation")  
    num = int("xyz")
```

```
except (ValueError, TypeError):
    print("Value or Type error occurred")
```

- Attempting operation
Value or Type error occurred
- Attempting operation
It raises a ValueError
- Value or Type error occurred
- It raises a TypeError

?

Q21:

What is the output of the following code?

```
try:
    print("Performing division")
    result = 10 / "2"
except TypeError:
    print("TypeError caught")
except ZeroDivisionError:
    print("ZeroDivisionError caught")
```

- Performing division
TypeError caught
- Performing division
ZeroDivisionError caught
- Performing division
It raises a ValueError
- It raises a ZeroDivisionError

?

Q22:

What is the output of the following code?

```
try:
    print("Processing data")
    data = [1, 2, 3]
    print(data[5])
except (IndexError, KeyError):
    print("Index or Key error occurred")
```

- Processing data
Index or Key error occurred
- Processing data
It raises an IndexError

- Index or Key error occurred

- Processing data

It raises a KeyError

?

Q23:

What is the output of the following code?

```
try:  
    print("Opening resource")  
    x = 10 / 0  
except (ZeroDivisionError, TypeError):  
    print("Caught ZeroDivisionError or TypeError")
```

- Opening resource
Caught ZeroDivisionError or TypeError
- Opening resource
It raises a ZeroDivisionError
- Caught ZeroDivisionError or TypeError
- It raises an error before printing

?

Q24:

What is the output of the following code?

```
try:  
    print("Opening resource")  
    x = int("not_an_int")  
except (ValueError, KeyError):  
    print("ValueError or KeyError caught")
```

- Opening resource
ValueError or KeyError caught
- Opening resource
It raises a ValueError
- ValueError or KeyError caught
- It raises an error before printing

?

Q25:

What is the output of the following code?

```
try:  
    print("Attempting operations")  
    x = int("10")  
    y = 10 / x
```

```
z = undefined_variable
except ZeroDivisionError:
    print("ZeroDivisionError caught")
except NameError:
    print("NameError caught")
except Exception:
    print("General exception caught")
```

- Attempting operations
ZeroDivisionError caught
- Attempting operations
NameError caught
- Attempting operations
General exception caught
- Attempting operations
It raises an AttributeError

❓ Q26:

What is the output of the following code?

```
try:
    print("Trying to convert")
    num = int("abc")
except ZeroDivisionError:
    print("ZeroDivisionError")
except (ValueError, TypeError):
    print("ValueError or TypeError")
except Exception:
    print("General Exception")
```

- Trying to convert
ZeroDivisionError
- Trying to convert
ValueError or TypeError
- Trying to convert
General Exception
- It raises a ValueError

❓ Q27:

What is the output of the following code?

```
try:
    print("Trying operation")
    x = "10" + 5
except TypeError:
```

```
print("TypeError caught")
except Exception:
    print("General exception")
```

- Trying operation
TypeError caught
- Trying operation
General exception
- TypeError caught
- It raises a TypeError before printing

❓ Q28:

What is the output of the following code? (Assume user inputs 0)

```
try:
    print("Processing input")
    value = int(input("Enter a number: "))
    result = 10 / value
except (ValueError, ZeroDivisionError):
    print("Invalid input or division by zero")
```

- Processing input
Invalid input or division by zero
- Processing input
It raises a ZeroDivisionError
- Invalid input or division by zero
- Processing input
It waits for input indefinitely

❓ Q29:

What is the output of the following code?

```
try:
    print("Attempting operations")
    x = int("10")
    y = 10 / x
    z = [1, 2, 3][5]
except IndexError:
    print("IndexError caught")
except ZeroDivisionError:
    print("ZeroDivisionError caught")
except Exception:
    print("General Exception")
```

- Attempting operations
IndexError caught
- Attempting operations
ZeroDivisionError caught
- Attempting operations
General Exception
- Attempting operations
It raises an IndexError

❓ Q30:

What is the output of the following code?

```
try:  
    print("Attempting operations")  
    x = int("10")  
    y = 10 / x  
    z = undefined_var  
except ZeroDivisionError:  
    print("ZeroDivisionError caught")  
except NameError:  
    print("NameError caught")  
except Exception:  
    print("General Exception")
```

- Attempting operations
ZeroDivisionError caught
- Attempting operations
NameError caught
- Attempting operations
General Exception
- Attempting operations
It raises a NameError

❓ Q31:

What is the output of the following code?

```
try:  
    print("Attempting division")  
    x = 10 / 2  
except ZeroDivisionError:  
    print("ZeroDivisionError caught")  
else:  
    print("Division successful")
```

- Attempting division

ZeroDivisionError caught

- Attempting division
- Division successful
- Division successful
- Attempting division

?

Q32:

What is the output of the following code?

```
try:  
    print("Starting process")  
    x = 10 / "2"  
except ZeroDivisionError:  
    print("ZeroDivisionError caught")  
except TypeError:  
    print("TypeError caught")  
finally:  
    print("Process ended")
```

- Starting process
 ZeroDivisionError caught
 Process ended
- Starting process
 TypeError caught
 Process ended
- Starting process
 Process ended
- It raises a TypeError without printing

?

Q33:

What is the output of the following code?

```
try:  
    print("Try block")  
    raise ValueError("Error message")  
except ValueError as ve:  
    print("Caught ValueError:", ve)  
    raise  
except Exception:  
    print("General Exception")  
finally:  
    print("Finally block")
```

- Try block
 Caught ValueError: Error message

- Finally block
- Try block
 - Caught ValueError: Error message
 - It raises a ValueError
 - Finally block
- Caught ValueError: Error message
- Finally block
- It raises a ValueError without printing anything

?

Q34:

What is the output of the following code?

```
try:  
    print("Attempting multiple operations")  
    a = [1, 2, 3]  
    print(a[5])  
    x = 10 / 0  
except IndexError:  
    print("IndexError caught")  
except ZeroDivisionError:  
    print("ZeroDivisionError caught")
```

- Attempting multiple operations
IndexError caught
ZeroDivisionError caught
- Attempting multiple operations
IndexError caught
- Attempting multiple operations
ZeroDivisionError caught
- It raises a ZeroDivisionError

?

Q35:

What is the output of the following code?

```
try:  
    print("Starting operations")  
    x = int("10")  
    y = 10 / x  
    z = [1, 2, 3][5]  
except (ZeroDivisionError, IndexError):  
    print("ZeroDivisionError or IndexError caught")  
finally:  
    print("Finalizing operations")
```

- Starting operations

ZeroDivisionError or IndexError caught

- Starting operations
 - It raises an IndexError
- ZeroDivisionError or IndexError caught
 - Finalizing operations
- Starting operations
 - ZeroDivisionError or IndexError caught
 - Finalizing operations

?

Q36:

What is the output of the following code?

```
try:  
    print("Initializing")  
    x = 5 / 0  
except ZeroDivisionError:  
    print("Caught ZeroDivisionError")  
except Exception:  
    print("Caught Exception")  
finally:  
    print("Cleanup completed")
```

- Initializing
 - Caught ZeroDivisionError
- Initializing
 - Caught ZeroDivisionError
 - Cleanup completed
- Caught ZeroDivisionError
 - Cleanup completed
- Initializing
 - Cleanup completed

?

Q37:

What is the output of the following code?

```
try:  
    print("Processing data")  
    data = [1, 2, 3]  
    print("Data length:", len(data))  
except Exception:  
    print("An error occurred")  
else:  
    print("No errors encountered")
```

- Processing data
Data length: 2
- Processing data
Data length: 3
No errors encountered
- Processing data
No errors encountered
- Processing data
An error occurred

?

Q38:

What is the output of the following code?

```
try:  
    print("Executing try block")  
    x = 5 + 5  
except ZeroDivisionError:  
    print("Division by zero")  
else:  
    print("No exceptions")
```

- Executing try block
Division by zero
- Executing try block
No exceptions
- No exceptions
- Executing try block
It raises an error

?

Q39:

What is the output of the following code?

```
try:  
    print("Executing try block")  
    x = 10 / 2  
except ZeroDivisionError:  
    print("Division by zero")  
else:  
    print("Division successful")
```

- Executing try block
Division successful
- Executing try block
Division by zero

- Division successful
- Executing try block

❓ Q40:

What is the output of the following code?

```
try:  
    print("Try block")  
    x = 10 + 5  
except Exception:  
    print("Exception caught")  
else:  
    print("Else block")  
finally:  
    print("Finally block")
```

- Try block
Else block
Finally block
- Try block
Exception caught
Finally block
- Try block
Finally block
- Exception caught
Finally block

❓ Q41:

What is the output of the following code?

```
try:  
    print("Attempting conversion")  
    num = int("123")  
except ValueError:  
    print("Conversion failed")  
else:  
    print("Conversion succeeded")
```

- Attempting conversion
Conversion failed
- Attempting conversion
Conversion succeeded
- Conversion succeeded
- It raises an error

?

Q42:

What is the output of the following code?

```
try:  
    print("Start")  
    x = 5 + 5  
except Exception:  
    print("Exception occurred")  
else:  
    print("No exceptions")  
finally:  
    print("End of block")
```

- Start
No exceptions
End of block
- Start
Exception occurred
End of block
- Start
End of block
- Start
It raises an exception
End of block

?

Q43:

What is the output of the following code?

```
try:  
    print("Attempting safe operation")  
    x = 10 + 5  
except ZeroDivisionError:  
    print("ZeroDivisionError")  
else:  
    print("Operation successful")
```

- Attempting safe operation
ZeroDivisionError
- Attempting safe operation
Operation successful
- Operation successful
- It raises an exception

?

Q44:

What is the output of the following code?

```
try:  
    print("Attempting to parse")  
    num = int("100")  
except ValueError:  
    print("ValueError caught")  
else:  
    print("Parsing successful")
```

- Attempting to parse
ValueError caught
- Attempting to parse
Parsing successful
- Parsing successful
- It raises an error

❓ Q45:

What is the output of the following code?

```
try:  
    print("Processing data")  
    data = [1, 2, 3]  
    print("Data accessed:", data[1])  
except IndexError:  
    print("IndexError caught")  
else:  
    print("Data processed successfully")  
finally:  
    print("Cleanup done")
```

- Processing data
Data accessed: 2
Data processed successfully
Cleanup done
- Processing data
Data accessed: 2
Cleanup done
- Data processed successfully
Cleanup done
- Processing data
Cleanup done

❓ Q46:

What is the output of the following code?

```
try:  
    print("Opening file")  
    f = open("non_existent_file.txt", "r")  
except FileNotFoundError:  
    print("File not found")  
finally:  
    print("Closing resources")
```

- Opening file
File not found
- Opening file
Closing resources
- Opening file
File not found
Closing resources
- Closing resources

?

Q47:

What is the output of the following code?

```
try:  
    print("Start")  
    raise ValueError("An error occurred")  
except ValueError as ve:  
    print("Caught an exception:", ve)  
finally:  
    print("End")
```

- Start
End
- Start
Caught an exception: An error occurred
End
- Start
It raises a ValueError
- Caught an exception: An error occurred
End

?

Q48:

What is the output of the following code?

```
try:  
    print("Outer try")  
    try:  
        print("Inner try")  
        y = 1 / 0  
    except ZeroDivisionError:  
        print("Inner except")  
    finally:  
        print("Inner finally")  
except Exception:  
    print("Outer except")  
finally:  
    print("Outer finally")
```

- Outer try

Inner try

Inner except

Inner finally

Outer finally

- Outer try

Inner try

Inner except

Outer except

Outer finally

- Outer try

Inner try

Inner except

Inner finally

Outer except

Outer finally

- Outer try

Inner try

Inner except

Outer except

Outer finally

?

Q49:

What is the output of the following code?

```
def test():  
    try:  
        return "Try block"  
    finally:  
        print("Finally block")  
  
result = test()
```

```
print(result)
```

- Finally block
- Try block
- Try block
- Finally block
- Finally block Try block
- Try block Finally block

?

Q50:

What is the output of the following code?

```
try:  
    print("Inside try")  
    raise KeyError("Missing key")  
except KeyError:  
    print("Caught KeyError")  
finally:  
    print("Executing finally block")
```

- Inside try
 Caught KeyError
- Inside try
 Caught KeyError
 Executing finally block
- Caught KeyError
 Executing finally block
- Inside try
 Executing finally block

?

Q51:

What is the output of the following code?

```
try:  
    print("Begin")  
    x = 10 / 0  
except ZeroDivisionError:  
    print("Division error")  
finally:  
    print("Cleanup actions")
```

- Begin
 Division error

- Begin
Division error
Cleanup actions
- Division error
Cleanup actions
- Begin
Cleanup actions

?

Q52:

What is the output of the following code?

```
try:  
    print("Start")  
    x = 10 / 2  
    y = int("abc")  
except ZeroDivisionError:  
    print("ZeroDivisionError")  
except ValueError:  
    print("ValueError")  
else:  
    print("No exceptions")  
finally:  
    print("End")
```

- Start
No exceptions
End
- Start
ValueError
End
- Start
ZeroDivisionError
End
- Start
It raises a ValueError
End

?

Q53:

What is the output of the following code?

```
def func():  
    try:  
        print("Try block")  
        return "Returning from try"  
    except:
```

```
    print("Except block")
finally:
    print("Finally block")

result = func()
print(result)
```

- Try block
Finally block
Returning from try
- Try block
Except block
Finally block
Returning from try
- Finally block
Returning from try
- Try block
It raises an exception

?

Q54:

What is the output of the following code?

```
try:
    print("Start")
    x = 1 / 0
except ZeroDivisionError:
    print("ZeroDivisionError caught")
    raise
finally:
    print("Finally block")
```

- Start
ZeroDivisionError caught
Finally block
- Start
ZeroDivisionError caught
It raises a ZeroDivisionError
Finally block
- ZeroDivisionError caught
Finally block
- Start
It raises a ZeroDivisionError

?

Q55:

What is the output of the following code?

```
try:  
    print("Accessing resource")  
    f = open("sample.txt", "r")  
except FileNotFoundError:  
    print("File not found")  
finally:  
    print("Releasing resource")
```

- Accessing resource
Releasing resource
- Accessing resource
File not found
Releasing resource
- Releasing resource
- Accessing resource
It raises a FileNotFoundError

?

Q56:

What is the output of the following code?

```
try:  
    print("Start")  
    x = 1 / 0  
except ZeroDivisionError:  
    print("ZeroDivisionError caught")  
finally:  
    print("Executing finally")
```

- Start
ZeroDivisionError caught
- Start
ZeroDivisionError caught
Executing finally
- Start
Executing finally
- It raises a ZeroDivisionError

?

Q57:

What is the output of the following code?

```
def test():  
    try:  
        return "Try block"
```

```
    finally:  
        return "Finally block"  
  
result = test()  
print(result)
```

- Try block
- Finally block
- It raises an error
- Try block Finally block

❓ Q58:

What is the output of the following code?

```
try:  
    print("Try block")  
    x = 5 / 0  
except ZeroDivisionError:  
    print("Caught ZeroDivisionError")  
except Exception:  
    print("Caught Exception")  
finally:  
    print("Executing finally")
```

- Try block
Caught ZeroDivisionError
- Try block
Caught ZeroDivisionError
Executing finally
- Try block
Caught Exception
Executing finally
- It raises a ZeroDivisionError before printing

❓ Q59:

What is the output of the following code?

```
try:  
    print("Starting process")  
    raise ValueError("Invalid value")  
except ValueError as ve:  
    print("Caught an exception:", ve)  
finally:  
    print("Process ended")
```

- Starting process
Caught an exception: Invalid value
- Caught an exception: Invalid value
Process ended
- Starting process
Process ended
- Process ended

?

Q60:

What is the output of the following code?

```
try:  
    print("Attempting operation")  
    x = int("not_an_int")  
except Exception:  
    print("General exception caught")  
except ValueError:  
    print("ValueError caught")
```

- Performing operations
ValueError caught
- Performing operations
General exception caught
- ValueError caught
- It raises a SyntaxError

?

Q61:

What is the output of the following code?

```
try:  
    print("Attempting division")  
    x = 10 / 2  
except ZeroDivisionError:  
    print("ZeroDivisionError caught")  
else:  
    print("Division successful")
```

- Attempting division
ZeroDivisionError caught
- Attempting division
Division successful
- Division successful
- Attempting division

?

Q62:

What is the output of the following code?

```
try:  
    print("Starting process")  
    x = 10 / "2"  
except ZeroDivisionError:  
    print("ZeroDivisionError caught")  
except TypeError:  
    print("TypeError caught")  
finally:  
    print("Process ended")
```

- Starting process
ZeroDivisionError caught
Process ended
- Starting process
TypeError caught
Process ended
- Starting process
Process ended
- It raises a TypeError without printing

?

Q63:

What is the output of the following code?

```
try:  
    print("Try block")  
    raise ValueError("Error message")  
except ValueError as ve:  
    print("Caught ValueError:", ve)  
    raise  
except Exception:  
    print("General Exception")  
finally:  
    print("Finally block")
```

- Try block
Caught ValueError: Error message
Finally block
- Try block
Caught ValueError: Error message
It raises a ValueError
Finally block

- Caught ValueError: Error message
Finally block
- It raises a ValueError without printing anything

?

Q64:

What is the output of the following code?

```
try:  
    print("Starting computation")  
    x = "10" / 2  
except Exception:  
    print("General Exception caught")  
except TypeError:  
    print("TypeError caught")
```

- Starting computation
General Exception caught
- Starting computation
TypeError caught
- It raises a SyntaxError
- TypeError caught

?

Q65:

What is the output of the following code?

```
try:  
    print("Outer try")  
    try:  
        print("Inner try")  
        x = 1 / 0  
    except ValueError:  
        print("Inner ValueError")  
except ZeroDivisionError:  
    print("Outer ZeroDivisionError")
```

- Outer try
Inner try
Outer ZeroDivisionError
- Outer try
Inner try
Inner ValueError
- Outer try
Inner try
Inner ValueError

Outer ZeroDivisionError

- Outer try
 - Inner try
 - It raises a ZeroDivisionError

?

Q66:

What is the output of the following code?

```
try:  
    print("Outer try")  
    try:  
        print("Inner try")  
        y = 1 / 0  
    except TypeError:  
        print("Inner TypeError")  
except ZeroDivisionError:  
    print("Outer ZeroDivisionError")
```

- Outer try
 - Inner try
 - Inner TypeError
- Outer try
 - Inner try
 - Outer ZeroDivisionError
- Outer try
 - Inner TypeError
 - Outer ZeroDivisionError
- It raises an error

?

Q67:

Which of the following exceptions is the base class for all built-in exceptions except for system-exiting exceptions?

- BaseException
- Exception
- RuntimeError
- ArithmeticError

?

Q68:

Which of the following is the correct order from the base class to the most specific exception?

- Exception -> ArithmeticError -> ZeroDivisionError
- ZeroDivisionError -> ArithmeticError -> Exception
- ArithmeticError -> ZeroDivisionError -> Exception
- BaseException -> Exception -> ZeroDivisionError

?

Q69:

What is the output of the following code?

```
try:  
    print("Executing try block")  
    raise KeyboardInterrupt  
except Exception:  
    print("Caught Exception")  
except BaseException:  
    print("Caught BaseException")
```

- Executing try block
Caught Exception
- Executing try block
Caught BaseException
- Executing try block
It raises a KeyboardInterrupt
- Caught BaseException

?

Q70:

What is the output of the following code?

```
try:  
    print("Performing operations")  
    x = int("not_an_int")  
except Exception:  
    print("General exception caught")  
except ValueError:  
    print("ValueError caught")
```

- Performing operations
ValueError caught
- Performing operations
General exception caught
- ValueError caught
- It raises a SyntaxError

?

Q71:

What is the output of the following code?

```
try:  
    print("Attempting operation")  
    x = int("abc")
```

```
except Exception:  
    print("Caught Exception")  
except ValueError:  
    print("Caught ValueError")
```

- Attempting operation
Caught Exception
- Attempting operation
Caught ValueError
- It raises a SyntaxError
- It raises a ValueError without printing

?

Q72:

What is the output of the following code?

```
try:  
    print("Attempting multiple operations")  
    a = [1, 2, 3]  
    print(a[5])  
    x = 10 / 0  
except IndexError:  
    print("IndexError caught")  
except ZeroDivisionError:  
    print("ZeroDivisionError caught")
```

- Attempting multiple operations
IndexError caught
ZeroDivisionError caught
- Attempting multiple operations
IndexError caught
- Attempting multiple operations
ZeroDivisionError caught
- It raises a ZeroDivisionError

?

Q73:

What is the output of the following code?

```
try:  
    print("Starting operations")  
    x = int("10")  
    y = 10 / x  
    z = [1, 2, 3][5]  
except (ZeroDivisionError, IndexError):  
    print("ZeroDivisionError or IndexError caught")  
finally:
```

```
print("Finalizing operations")
```

- Starting operations
ZeroDivisionError or IndexError caught
- Starting operations
It raises an IndexError
- ZeroDivisionError or IndexError caught
Finalizing operations
- Starting operations
ZeroDivisionError or IndexError caught
Finalizing operations

❓ Q74:

What is the output of the following code?

```
try:  
    print("Attempting risky computation")  
    x = 10 / "2"  
except TypeError:  
    pass  
print("Computation done")
```

- Attempting risky computation
Computation done
- Computation done
- It raises a TypeError
- Attempting risky computation
It raises a TypeError

❓ Q75:

What is the output of the following code?

```
try:  
    print("Initializing")  
    x = 5 / 0  
except ZeroDivisionError:  
    print("Caught ZeroDivisionError")  
except Exception:  
    print("Caught Exception")  
finally:  
    print("Cleanup completed")
```

- Initializing
Caught ZeroDivisionError

- Initializing
Caught ZeroDivisionError
Cleanup completed
 - Caught ZeroDivisionError
Cleanup completed
 - Initializing
Cleanup completed
-

Licensed for: WBS Training AG
PYDASC71e-13

✓ The Answers

Q#	Answer	Explanation of the answer
1	B	Division by 2 is valid, prints both "Attempting division" and "Result is 5.0".
2	B	File not found, printed, then re-raises FileNotFoundError after printing.
3	B	Division by zero not caught by ValueError, so it raises ZeroDivisionError after printing "Start".
4	A	ValueError caught by <code>except ValueError: pass</code> , then prints "Operation completed".
5	A	Exception raised and caught, prints "Starting process" then "Caught an exception: Custom error".
6	A	<code>undefined_variable</code> causes NameError, caught by bare except, prints both lines.
7	C	<code>10/0</code> raises ZeroDivisionError, caught by except block, prints "Cannot divide by zero".
8	A	Adding int and str raises TypeError, caught by Exception, prints "Attempting operation" then "An exception occurred".
9	A	Accessing non-existent attribute raises AttributeError, caught by except block, prints both.
10	A	<code>int("not a number")</code> raises ValueError, caught and passed, then "Operation completed".
11	A	<code>1/"a"</code> raises TypeError, caught by bare except, prints "Running code" and "An error occurred".
12	B	<code>d['c']</code> raises KeyError, caught by KeyError except, prints both.
13	A	<code>1/0</code> raises ZeroDivisionError, caught and passed, then "Operation completed".
14	B	<code>10/0</code> not caught by ValueError, caught by Exception block, prints "General Exception caught".
15	B	<code>KeyboardInterrupt</code> caught by except block, prints "KeyboardInterrupt caught".
16	B	<code>float("NaN")</code> succeeds, prints "Attempting conversion" and "Conversion successful".
17	A	<code>10/"2"</code> raises TypeError, caught and passed, then "Computation done".
18	A	<code>lst[5]</code> raises IndexError, caught by first except, prints both.
19	A	<code>int("abc")</code> raises ValueError, caught by first except, prints both.
20	A	<code>int("xyz")</code> raises ValueError, caught by (ValueError,TypeError), prints both.
21	A	<code>10/"2"</code> raises TypeError, caught by TypeError except, prints both.
22	A	<code>data[5]</code> raises IndexError, caught by (IndexError,KeyError), prints both.
23	A	<code>10/0</code> raises ZeroDivisionError, caught by combined except, prints both.

Q#	Answer	Explanation of the answer
24	A	int("not_an_int") raises ValueError, caught by (ValueError,KeyError), prints both.
25	B	undefined_variable causes NameError, caught by NameError except.
26	B	int("abc") raises ValueError, caught by (ValueError,TypeError), prints both lines.
27	A	"10"+5 raises TypeError, caught by TypeError except, prints both.
28	A	Input=0 causes ZeroDivisionError, caught by combined except, prints both.
29	A	[1,2,3][5] raises IndexError, caught by IndexError except, prints both.
30	B	undefined_var raises NameError, caught by NameError except.
31	B	10/2 is fine, no error, else prints "Division successful".
32	B	10/"2" raises TypeError, caught by second except, finally executes, prints all three lines.
33	A	ValueError caught and re-raised, finally executes, prints try, caught line, finally block before error.
34	B	[1,2,3][5] raises IndexError first, caught, prints both, division by zero not reached.
35	D	Accessing [1,2,3][5] raises IndexError caught by combined except, then finally executes.
36	B	5/0 raises ZeroDivisionError, caught, prints those lines, then finally block.
37	B	len(data)=3, no error, prints them and "No errors encountered".
38	B	No error, else executes "No exceptions".
39	A	10/2 no error, prints "Division successful".
40	A	No error, else and finally execute.
41	B	int("123") success, prints "Conversion succeeded".
42	A	No error, else and finally execute.
43	B	No error, else prints "Operation successful".
44	B	int("100") success, "Parsing successful".
45	A	No error, else and finally execute.
46	C	non_existent_file raises FileNotFoundError, caught, then finally executes.
47	B	ValueError raised and caught, then finally.
48	A	Inner ZeroDivisionError caught by inner except, inner finally, no outer except triggered, outer finally.
49	A	finally executes before returning try block value, prints "Finally block" then "Try block" as return value.
50	B	KeyError caught, then finally executes.

Q#	Answer	Explanation of the answer
51	B	ZeroDivisionError caught, then finally executes.
52	B	int("abc") raises ValueError, caught by second except, then finally executes.
53	A	Returns from try, finally prints, returns "Returning from try".
54	B	ZeroDiv caught, re-raised, finally executes.
55	B	FileNotFoundException caught, then finally executes.
56	B	ZeroDiv caught, then finally.
57	B	finally return overrides try return, prints finally block and returns "Finally block".
58	B	ZeroDiv caught, then finally.
59	A	ValueError caught, then finally.
60	D	More specific ValueError should be before Exception, causes SyntaxError.
61	B	10/2 no error, else prints "Division successful".
62	B	10/"2" TypeError, caught, then finally.
63	B	ValueError caught, re-raised, finally executes.
64	C	More specific after general causes SyntaxError.
65	A	Outer try Inner try Outer ZeroDivisionError
66	B	ZeroDivisionError caught by outer block.
67	B	Exception is the base class for most built-ins.
68	A	Exception -> ArithmeticError -> ZeroDivisionError correct order.
69	B	KeyboardInterrupt caught by BaseException block, not by Exception.
70	D	More general Exception before ValueError causes SyntaxError.
71	C	More general Exception before ValueError causes SyntaxError.
72	B	IndexError caught first, ZeroDivisionError not reached.
73	D	IndexError caught by combined except, then finally.
74	A	TypeError caught and passed, prints "Computation done".
75	B	ZeroDiv caught, then finally executes.

2.3 Modules

Required Pre-Knowledge

- Understanding basic Python math functions (sqrt, factorial, pow, log, ceil, floor, fabs, gcd, exp, sin)
 - Understanding Python's import system (importing modules, using aliases, from-import syntax, importing specific functions, and using '*')
 - Knowledge of Python's `__pycache__` directory, what it stores, and how Python manages compiled bytecode
-

Quiz

It contains 20 questions. Please answer the questions on separate paper or in an Excel file, and then compare your answers with those in the answers section after the quiz.

Q1:

What is the output of the following code?

```
import math
result = math.sqrt(16)
print(result)
```

- 14
- 4
- 8
- 16

Q2:

What does the following code output?

```
import math
print(math.factorial(5))
```

- 60
- 120
- 24
- 720

Q3:

What is the output of the following code?

```
import math  
print(math.pow(3, 3))
```

- 6
- 9
- 27
- 81

?

Q4:

What does the following code output?

```
import math  
print(math.log(math.e))
```

- 0
- 1
- 2
- e

?

Q5:

What is the result of the following code?

```
import math  
print(math.ceil(4.2))
```

- 4
- 4.2
- 5
- 6

?

Q6:

What does the following code output?

```
import math  
print(math.floor(7.8))
```

- 7
- 8
- 7.8

- 6

?

Q7:

What is the output of the following code?

```
import math
print(math.fabs(-5))
```

- -5
- 0
- 5
- 10

?

Q8:

What does the following code output?

```
import math
print(math.gcd(48, 18))
```

- 6
- 12
- 18
- 24

?

Q9:

What is the result of the following code?

```
import math
print(math.exp(2))
```

- 7
- 8
- 9
- Approximately 7.389

?

Q10:

What does the following code output?

```
import math
print(math.sin(math.pi / 2))
```

- 0
- 0.5
- 1
- Undefined

❓ Q11:

What is the correct way to import the `math` module in Python?

```
# Which import statement correctly imports the math module?
```

- import math
- import Math
- from math import *
- include math

❓ Q12:

How do you import the `math` module with the alias `m`?

```
# Select the correct import statement
```

- import math as m
- import math alias m
- from math import as m
- import m as math

❓ Q13:

Given the import statement `import math`, how do you call the `sqrt` function from the `math` module?

```
import math
result = ?
```

- sqrt(16)
- math.sqrt(16)
- sqrt math(16)
- math-sqrt(16)

?

Q14:

How do you import only the `sqrt` function from the `math` module?

```
# Select the correct import statement
```

- `import math.sqrt`
- `from math import sqrt`
- `import sqrt from math`
- `from math import * sqrt`

?

Q15:

After importing `sqrt` using `from math import sqrt`, how do you call it to compute the square root of 25?

```
from math import sqrt
result = ?
```

- `math.sqrt(25)`
- `sqrt(25)`
- `sqrt.math(25)`
- `math-sqrt(25)`

?

Q16:

What does the following import statement do?

```
from math import *
```

- Imports nothing
- Imports all functions from the math module
- Imports the math module as a wildcard
- Imports only the `sqrt` function

?

Q17:

What is a potential downside of using `from math import *`?

```
from math import *
```

- It doesn't import any functions
- It makes all math functions unavailable

- It can lead to namespace pollution
- It only imports private functions

?

Q18:

What is the purpose of the `__pycache__` directory in a Python project?

```
# What does __pycache__ store?
```

- Source code files
- Compiled bytecode files
- Documentation files
- Configuration files

?

Q19:

How can you prevent Python from generating `.pyc` files in the `__pycache__` directory?

```
# What environment variable can be set to disable .pyc file generation?
```

- PYTHONPATH
- PYTHONDONTWRITEBYTECODE
- PYTHONNOCACHE
- PYTHONIGNORECACHE

?

Q20:

If the `__pycache__` directory is deleted, what happens when you run a Python program again?

```
# What occurs if __pycache__ is missing when running a Python script?
```

- Python cannot run the script
- Python regenerates the `__pycache__` directory
- Python runs the script without caching bytecode
- The script runs with errors

✓ The Answers

Question Nr.	Answer	Explanation of the answer
1	B	$\sqrt{16} = 4$
2	B	$\text{factorial}(5) = 120$
3	C	$3^3 = 27$
4	B	$\log(e) = 1$
5	C	$\text{ceil}(4.2) = 5$
6	A	$\text{floor}(7.8) = 7$
7	C	$\text{fabs}(-5) = 5$
8	A	$\text{gcd}(48,18) = 6$
9	D	$\exp(2) \approx 7.389$
10	C	$\sin(\pi/2) = 1$
11	A	Correct import is 'import math'
12	A	'import math as m' for alias
13	B	Access with <code>math.sqrt(16)</code>
14	B	'from math import sqrt'
15	B	Call directly <code>sqrt(25)</code>
16	B	Imports all public names from math
17	C	Leads to namespace pollution
18	B	pycache stores compiled bytecode
19	B	Set <code>PYTHONONTWRITEBYTECODE</code> to disable .pyc
20	B	Python will automatically recreate it and regenerate the .pyc

2.4 Built-In Functions

Required Pre-Knowledge

- Understanding of Python built-in functions and their purposes.
 - Familiarity with Python's basic data types (`int`, `float`, `str`, `list`, `tuple`, `dict`, `set`) and their boolean evaluations.
 - Knowledge of function decorators like `@classmethod`, `@staticmethod`, and `@property`.
 - Understanding Python iteration protocols, including `iter()`, `next()`, and `enumerate()`.
 - Awareness of `eval()`, `exec()` functions and their implications.
 - Ability to handle exceptions raised by built-in functions (e.g., `ValueError`, `AttributeError`).
 - Basic understanding of Unicode, character encoding, and special string representations.
 - Familiarity with Python's object model (classes, instances, inheritance, `super()`).
 - Knowledge of slicing, memory views, and mutable vs. immutable sequences.
 - Understanding the difference between functions like `all()`, `any()`, `filter()`, `map()`, `sum()`, `sorted()`.
-

Quiz

It contains 80 questions. Please answer the questions on separate paper or in an Excel file, and then compare your answers with those in the answers section after the quiz.

Q1: `abs()`

What is the output of the following code?

```
number = -15
print(abs(number))
```

Options:

- -15
 - 0
 - 15
 - Error
-

Q2: `aiter()`

What does the following asynchronous code do?

```
import asyncio
```

```
async def main():
    async for char in aiter("ABC"):
        print(char)

asyncio.run(main())
```

Options:

- It prints each character immediately without delay
 - It raises an error because `aiter` is not used correctly
 - It creates an asynchronous iterator and prints each character
 - It does nothing
-

?

Q3: `all()`

```
values = [True, True, False]
print(all(values))
```

Options:

- `True`
 - `False`
 - It raises an error
 - `None`
-

?

Q4: `any()`

```
values = [False, False, True]
print(any(values))
```

Options:

- `True`
 - `False`
 - It raises an error
 - `None`
-

?

Q5: `ascii()`

```
text = "Café"
print(ascii(text))
```

Options:

- "Café"
 - 'Caf\xe9'
 - "Caf\\xe9"
 - It raises an error
-

❓ Q6: `bin()`

```
number = 10
print(bin(number))
```

Options:

- 10
 - 0b1010
 - 1010
 - It raises an error
-

❓ Q7: `bool()`

```
print(bool(0))
print(bool(1))
print(bool(""))
print(bool("Hello"))
```

Options:

- False True False True
 - True False True False
 - False True True False
 - True True False True
-

❓ Q8: `breakpoint()`

```
def test():
    breakpoint()
    print("After breakpoint")

test()
```

Options:

- It prints "After breakpoint" immediately
 - It triggers the debugger and pauses execution
 - It raises an error
 - It does nothing
-

❓ Q9: `bytearray()`

```
data = bytearray("hello", "utf-8")
data[0] = 72
print(data.decode())
```

Options:

- Hello
 - hello
 - Error
 - Hhello
-

❓ Q10: `bytes()`

```
data = bytes([50, 100, 76])
print(data)
```

Options:

- b'50,100,76'
 - b'2dL'
 - b'2dL'
 - It raises an error
-

❓ Q11: `callable()`

```
def func():
    pass

class MyClass:
    pass

obj1 = func
obj2 = MyClass()

print(callable(obj1))
print(callable(obj2))
```

Options:

- True True
 - True False
 - False True
 - False False
-

❓ Q12 : `chr()`

```
print(chr(65))
print(chr(97))
```

Options:

- A a
 - a A
 - 65 97
 - It raises an error
-

❓ Q13 : `classmethod()`

```
class MyClass:
    value = 10

    @classmethod
    def get_value(cls):
        return cls.value

print(MyClass.get_value())
```

Options:

- 10
- `cls.value`
- It raises an error
- `MyClass.value`

❓ Q14: `compile()`

```
code_str = "print('Compiled Code')"
code = compile(code_str, '<string>', 'exec')
exec(code)
```

Options:

- It prints `Compiled Code`
- It does nothing
- It raises an error
- It compiles but does not execute

❓ Q15: `complex()`

```
c = complex(2, 3)
print(c.real)
print(c.imag)
```

Options:

- 2 3
- 3 2
- 2.0 3.0
- Error

❓ Q16: `delattr()`

```
class Person:
    def __init__(self):
        self.name = "Alice"

p = Person()
print(p.name)
delattr(p, 'name')
print(hasattr(p, 'name'))
```

Options:

- Alice True
 - Alice False
 - Error
 - Alice
-

❓ Q17: `dict()` (Creating from keyword args)

```
data = dict(a=1, b=2, c=3)
print(data)
```

Options:

- {'a': 1, 'b': 2, 'c': 3}
 - {'a', 'b', 'c'}
 - ['a':1,'b':2,'c':3]
 - {'a':'1','b':'2','c':'3'}
-

❓ Q18: `dict()` (Creating from list of tuples)

```
items = [('a',1),('b',2),('c',3)]
d = dict(items)
print(d)
```

Options:

- [('a',1),('b',2),('c',3)]
 - {'a':1,'b':2,'c':3}
 - {'a':'1','b':'2','c':'3'}
 - It raises an error
-

❓ Q19: `dir()`

```
class MyClass:
    def __init__(self):
        self.attr1 = 1
        self.attr2 = 2

obj = MyClass()
print(dir(obj))
```

Options:

- It prints the values of `attr1` and `attr2`
 - It prints a list of attribute names of `obj`
 - It prints `['attr1', 'attr2']`
 - It raises an error
-

?

Q20: `divmod()`

```
result = divmod(-7, 3)
print(result)
```

Options:

- `(-3, 2)`
 - `(-2, 1)`
 - `(2, -1)`
 - `(-3, -2)`
-

?

Q21: `enumerate()`

```
fruits = ['apple', 'banana', 'cherry']
for index, fruit in enumerate(fruits, start=1):
    print(index, fruit)
```

Options:

- `0 apple 1 banana 2 cherry`
 - `1 apple 2 banana 3 cherry`
 - `apple banana cherry`
 - `apple 1 banana 2 cherry 3`
-

?

Q22: `eval()`

```
expr = "2 + 3 * 4"
result = eval(expr)
print(result)
```

Options:

- `14`
- `20`

- Error
 - "2 + 3 * 4"
-

❓ Q23 : `exec()`

```
code = """
def greet():
    print("Hello from exec!")

greet()
"""

exec(code)
```

Options:

- It defines `greet` but does not print
 - It raises an error
 - It prints `Hello from exec!`
 - It prints nothing
-

❓ Q24 : `filter()` (lambda even)

```
numbers = [1, 2, 3, 4, 5, 6]
even = filter(lambda x: x % 2 == 0, numbers)
print(list(even))
```

Options:

- [2,4,6]
 - [1,3,5]
 - [1,2,3,4,5,6]
 - []
-

❓ Q25 : `filter()` (filtering truthy)

```
values = [0, 1, False, True, None, '']
filtered = filter(None, values)
print(list(filtered))
```

Options:

- [0, False, None, '']

- [1, True]
 - []
 - [0, 1, False, True, None, '']
-

❓ Q26: `filter()` (is_even function)

```
def is_even(x):
    return x % 2 == 0

numbers = [1,2,3,4,5]
filtered = filter(is_even, numbers)
print(list(filtered))
```

Options:

- [1, 2, 3, 4, 5]
 - [2, 4]
 - [1, 3, 5]
 - []
-

❓ Q27: `float()`

```
value = "3.14159"
pi = float(value)
print(pi)
```

Options:

- "3.14159"
 - 3.14159
 - Error
 - 3
-

❓ Q28: `format()`

```
name = "Alice"
age = 30
print("Name: {}, Age: {}".format(name, age))
```

Options:

- Name: Alice Age: 30

- Name: {}, Age: {}
 - Name: Alice, Age: 30
 - Error
-

❓ Q29: `frozenset()`

```
s = frozenset([1,2,2,3])
print(s)
```

Options:

- {1,2,3}
 - [1,2,3]
 - frozenset({1,2,3})
 - frozenset([1,2,3])
-

❓ Q30: `getattr()`

```
class Person:
    name = "Bob"

p = Person()
print(getattr(p, 'name'))
print(getattr(p, 'age', 'Not Found'))
```

Options:

- Bob Not Found
 - Bob None
 - Bob Error
 - Error
-

❓ Q31: `globals()`

```
x = 10
y = 20

def func():
    z = 30
    print(globals())

func()
```

Options:

- `{'x':10, 'y':20, 'func':<function>}`
 - `{'z':30}`
 - `{'x':10, 'y':20, 'z':30}`
 - `{'func':<function>, 'z':30}`
-

❓ Q32 : `hasattr()`

```
class Car:  
    model="Sedan"  
  
c = Car()  
print(hasattr(c, 'model'))  
print(hasattr(c, 'color'))
```

Options:

- `True True`
 - `True False`
 - `False True`
 - `False False`
-

❓ Q33 : `hash()`

```
print(hash("Python"))  
print(hash(100))
```

Options:

- Two identical hash values
 - Different hash values
 - Error
 - `0 0`
-

❓ Q34 : `help()`

```
help(str)
```

Options:

- Prints string representation of `str`

- Shows help documentation for `str`
 - Error
 - Does nothing
-

❓ Q35 : `hex()`

```
number = 255
print(hex(number))
```

Options:

- '255'
 - '0xff'
 - 'FF'
 - 255
-

❓ Q36 : `id()`

```
a = 10
b = 10
print(id(a) == id(b))
```

Options:

- True
 - False
 - Error
 - None
-

❓ Q37 : `input()`

If user enters `Python`:

```
language = input("Enter a programming language: ")
print("You entered:", language)
```

Options:

- Enter a programming language: Python You entered: Python
- You entered: Python
- Enter a programming language:
- Python

❓ Q38 : `int()`

```
value = "123"  
number = int(value)  
print(number+10)
```

Options:

- "12310"
- 133
- 12310
- Error

❓ Q39 : `isinstance()`

```
num=5  
print(isinstance(num,int))  
print(isinstance(num,float))
```

Options:

- True True
- True False
- False True
- False False

❓ Q40 : `issubclass()`

```
class Animal: pass  
class Dog(Animal): pass  
  
print(issubclass(Dog,Animal))  
print(issubclass(Animal,Dog))
```

Options:

- True True
- True False
- False True
- False False

?

Q41 : `iter()`

```
numbers=[1,2,3]
it=iter(numbers)
print(next(it))
print(next(it))
print(next(it))
```

Options:

- 1 2 3
 - <iterator> <iterator> <iterator>
 - Error
 - 1 2
-

?

Q42 : `len()`

```
fruits=['apple','banana','cherry']
print(len(fruits))
```

Options:

- 2
 - 3
 - 4
 - Error
-

?

Q43 : `list()`

```
t=(1,2,3)
lst=list(t)
print(lst)
```

Options:

- (1,2,3)
 - [1,2,3]
 - ['1','2','3']
 - Error
-

?

Q44 : `locals()`

```
def func():
    x=10
    y=20
    print(locals())

func()
```

Options:

- `{'x':10, 'y':20}`
 - `{'x':10}`
 - `{'y':20}`
 - Error
-

❓ Q45 : `map()`

```
def square(x):
    return x*x

numbers=[1,2,3]
squared=map(square,numbers)
print(list(squared))
```

Options:

- `[1,4,9]`
 - `[1,2,3]`
 - `[2,3,4]`
 - Error
-

❓ Q46 : `max()` (numbers)

```
numbers=[10,20,30,40]
print(max(numbers))
```

Options:

- 10
 - 40
 - 30
 - Error
-

?

Q47 : `max()` (letters)

```
letters=['a','z','m']
print(max(letters))
```

Options:

- 'a'
 - 'z'
 - 'm'
 - Error
-

?

Q48 : `memoryview()` (hello)

Assume code from explanation:

(Similar to bytearray example)

This modifies the first byte of a bytearray `b'hello'` to `d`, resulting in `b'dello'`.

(No exact snippet given in final due to complexity, trusting original solution)

Options:

- `bytearray(b'dello')`
 - `bytearray(b'hello')`
 - Error
 - `bytearray(b'0ello')`
-

?

Q49 : `memoryview()` (abc)

```
data=bytearray(b'abc')
mv=memoryview(data)
mv[0]=100
print(data)
```

Options:

- `bytearray(b'dbc')`
 - `bytearray(b'abc')`
 - `bytearray(b'0bc')`
 - Error
-

?

Q50 : `min()` (values)

```
values=[3.14,2.71,1.41]  
print(min(values))
```

Options:

- 1.41
 - 2.71
 - 3.14
 - 0
-

❓ Q51 : **min()** (tuple)

```
values=(5,3,9,1)  
print(min(values))
```

Options:

- 1
 - 3
 - 5
 - 9
-

❓ Q52 : **next()**

```
numbers=iter([10,20,30])  
print(next(numbers))  
print(next(numbers))
```

Options:

- 10 30
 - 10 20
 - 20 30
 - Error
-

❓ Q53 : **next()** (with default)

```
it=iter([1,2])
print(next(it,'No more'))
print(next(it,'No more'))
print(next(it,'No more'))
```

Options:

- 1 1
 - 1 2 No more
 - 1 2 3
 - 1 2 2
-

?

Q54 : **object()**

```
obj=object()
print(obj)
```

Options:

- <object object at 0x...>
 - {}
 - None
 - Error
-

?

Q55 : **object()** (comparing two objects)

```
obj1=object()
obj2=object()
print(obj1 is obj2)
```

Options:

- True
 - False
 - Error
 - None
-

?

Q56 : **oct()**

```
number=8
print(oct(number))
```

Options:

- '10'
 - '0o10'
 - '8'
 - Error
-

❓ Q57 : **ord()** ('A')

```
char='A'
print(ord(char))
```

Options:

- 'A'
 - 65
 - 97
 - Error
-

❓ Q58 : **ord()** ('€')

```
print(ord('€'))
```

Options:

- 128
 - 8364
 - 8365
 - Error
-

❓ Q59 : **pow()** (2,3)

```
print(pow(2,3))
print(pow(2,3,3))
```

Options:

- 8 2
 - 8 1
 - 6 2
 - 8 0
-

❓ Q60 : **pow()** (3,4,5)

```
print(pow(3,4,5))
```

Options:

- 81
 - 1
 - 3
 - 0
-

❓ Q61 : **print()** (sep)

```
print(1,2,3,sep=' - ')
```

Options:

- 1 2 3
 - 1-2-3
 - 123
 - Error
-

❓ Q62 : **print()** (sep,end)

```
print("Hello","World",sep=' - ',end=' ! ')
```

Options:

- Hello World!
 - Hello-World!
 - Hello-World
 - HelloWorld!
-

❓ Q63 : **property()**

```
class Circle:  
    def __init__(self, radius):  
        self._radius = radius  
  
    @property  
    def radius(self):  
        return self._radius  
  
c = Circle(5)  
print(c.radius)  
c.radius = 10
```

Options:

- Prints 5 and updates to 10
- Prints 5 and raises error when updating
- Error before printing
- Prints 10

Q64 : range()

```
for i in range(1, 4):  
    print(i, end=' ')
```

Options:

- 1 2 3 4
- 0 1 2
- 1 2 3
- 1 2 3 4

Q65 : repr()

```
class Point:  
    def __init__(self, x, y):  
        self.x = x  
        self.y = y  
  
p = Point(1, 2)  
print(repr(p))
```

Options:

- Point(1, 2)

- `{'x':1,'y':2}`
 - `<__main__.Point object at 0x...>`
 - Error
-

?

Q66 : `set()`

```
numbers=[1,2,2,3,3,3]
unique=set(numbers)
print(unique)
```

Options:

- `[1,2,3]`
 - `{1,2,3}`
 - `{'1','2','3'}`
 - `set([1,2,3])`
-

?

Q67 : `setattr()`

```
class Book:
    pass

b=Book()
setattr(b,'title','Python Programming')
print(b.title)
```

Options:

- `'Python Programming'`
 - `Python Programming`
 - Error
 - `title`
-

?

Q68 : `slice()`

```
text="Hello, World!"
s=slice(7,12)
print(text[s])
```

Options:

- `World`

- World!
 - orld
 - Hello
-

❓ Q69 : **sorted()** (by length)

```
words=[ 'banana' , 'apple' , 'cherry' ]  
sorted_words=sorted(words, key=len)  
print(sorted_words)
```

Options:

- ['banana','apple','cherry']
 - ['apple','banana','cherry']
 - ['apple','cherry','banana']
 - ['cherry','apple','banana']
-

❓ Q70 : **sorted()** (dict keys)

```
d={'b':2, 'a':1, 'c':3}  
sorted_keys=sorted(d)  
print(sorted_keys)
```

Options:

- ['b','a','c']
 - ['a','b','c']
 - [1,2,3]
 - ['c','b','a']
-

❓ Q71 : **staticmethod()**

```
class Math:  
    @staticmethod  
    def add(a,b):  
        return a+b  
  
print(Math.add(3,4))
```

Options:

- 7

- 34
 - Error
 - add(3,4)
-

❓ Q72 : **str()**

```
number=100
text="Number is: "+str(number)
print(text)
```

Options:

- Number is: 100
 - Number is:100
 - Number is: '100'
 - Error
-

❓ Q73 : **sum()**

```
numbers=[1,2,3]
total=sum(numbers,10)
print(total)
```

Options:

- 6
 - 10
 - 16
 - -6
-

❓ Q74 : **super()**

```
class Parent:  
    def greet(self):  
        print("Hello from Parent")  
  
class Child(Parent):  
    def greet(self):  
        super().greet()  
        print("Hello from Child")  
  
c=Child()  
c.greet()
```

Options:

- Hello from Parent Hello from Child
- Hello from Child Hello from Parent
- Hello from Parent
- Hello from Child

❓ Q75 : `super()` (with arguments)

```
class Parent:  
    def greet(self, name):  
        print(f"Hello, {name} from Parent")  
  
class Child(Parent):  
    def greet(self, name):  
        super().greet(name)  
        print(f"Hello, {name} from Child")  
  
c=Child()  
c.greet("Diana")
```

Options:

- Hello, Diana from Parent Hello, Diana from Child
- Hello, Diana from Child Hello, Diana from Parent
- Hello, Diana from Parent
- Error

❓ Q76 : `tuple()`

```
lst=[1,2,3]
t=tuple(lst)
print(t)
```

Options:

- [1,2,3]
- (1,2,3)
- ['1','2','3']
- ('1,2,3')

❓ Q77 : **type()**

```
print(type(3.14))
print(type("Hello"))
```

Options:

- <class 'int'> <class 'str'>
- <class 'float'> <class 'str'>
- <class 'float'> <class 'int'>
- <class 'str'> <class 'str'>

❓ Q78 : **vars()** (inside function)

```
def func():
    a=1
    b=2
    print(vars())

func()
```

Options:

- {'a':1,'b':2}
- {'a':1,'b':2,'func':}
- {'name':'main',...}
- {'a':1}

❓ Q79 : **vars()** (global)

```
x=10
y=20
print(vars())
```

Options:

- {'x':10,'y':20}
- {'name':'main','x':10,'y':20,...}
- Prints nothing
- {'x':10}

❓ Q80 : **zip()**

```
names=['Alice', 'Bob', 'Charlie']
ages=[25, 30]
zipped=zip(names, ages)
print(list(zipped))
```

Options:

- [('Alice',25),('Bob',30),('Charlie',None)]
- [('Alice',25),('Bob',30)]
- [('Alice',25),('Bob',30),('Charlie',0)]
- Error

✓ The Answers

Q Nr.	Answer	Explanation
1	C	<code>abs(-15)=15</code>
2	C	<code>aiter creates async iterator</code>
3	B	<code>all([True,True,False])=False</code>
4	A	<code>any([False,False,True])=True</code>
5	B	<code>ascii("Café")='Caf\xe9'</code>
6	B	<code>bin(10)='0b1010'</code>
7	A	<code>bool(0)=False, bool(1)=True, bool("")=False, bool("Hello")=True</code>
8	B	<code>breakpoint enters debugger</code>
9	A	<code>bytearray 'hello' -> 'Hello' after data[0]=72</code>
10	B	<code>bytes([50,100,76])='2dL'</code>
11	B	<code>func is callable, obj2 not callable</code>
12	A	<code>chr(65)='A', chr(97)='a'</code>
13	A	<code>classmethod returns 10</code>
14	A	<code>compile+exec prints 'Compiled Code'</code>
15	C	<code>complex(2,3).real=2.0, imag=3.0</code>
16	B	<code>delattr removes name, hasattr now False</code>
17	A	<code>dict(a=1,b=2,c=3)={'a':1,'b':2,'c':3}</code>
18	B	<code>dict from list of tuples={'a':1,'b':2,'c':3}</code>
19	B	<code>dir(obj) prints list of attrs</code>
20	A	<code>divmod(-7,3)=(-3,2)</code>
21	B	<code>enumerate start=1 => 1 apple, 2 banana, 3 cherry</code>
22	A	<code>eval('2+3*4')=14</code>
23	C	<code>exec defines greet, calls it => prints 'Hello from exec!'</code>
24	A	<code>filter even => [2,4,6]</code>
25	B	<code>filter(None,...) keeps truthy => [1,True]</code>
26	B	<code>filter(is_even) => [2,4]</code>
27	B	<code>float("3.14159")=3.14159</code>
28	C	<code>format inserts values => 'Name: Alice Age: 30'</code>
29	C	<code>frozenset removes duplicates => frozenset({1,2,3})</code>

Q Nr.	Answer	Explanation
30	A	getattr(p,'name')='Bob',getattr(p,'age','Not Found')='Not Found'
31	A	globals() has x,y,func, not z (local)
32	B	hasattr(c,'model')=True,'color'=False
33	B	hash("Python") and hash(100) differ
34	B	help(str) shows doc for str
35	B	hex(255)='0xff'
36	A	same object id for small int => True
37	B	input user 'Python' => "You entered: Python"
38	B	int("123")+10=133
39	B	isinstance(5,int)=True,isinstance(5,float)=False
40	B	Dog subclass of Animal=True,Animal of Dog=False
41	A	iter+next => 1,2,3
42	B	len(['apple','banana','cherry'])=3
43	B	list((1,2,3))=[1,2,3]
44	A	locals() in func {'x':10,'y':20}
45	A	map(square,[1,2,3])=[1,4,9]
46	B	max(10,20,30,40)=40
47	B	max(['a','z','m'])='z'
48	A	memoryview changes first char => 'dello' (assumed)
49	A	memoryview('abc'):mv[0]=100 => 'dbc'
50	A	min(3.14,2.71,1.41)=1.41 => lowest 1.41 is approx 1.41.
51	A	min(5,3,9,1)=1
52	B	next => 10,20
53	B	next with default => 1,2,'No more'
54	A	object() prints <object object at ...>
55	B	obj1 is obj2? False
56	B	oct(8)='0o10'
57	B	ord('A')=65
58	B	ord('€')=8364
59	A	pow(2,3)=8,pow(2,3,3)=2

Q Nr.	Answer	Explanation
60	B	<code>pow(3,4,5)=1</code>
61	B	<code>print(1,2,3,sep='-')='1-2-3'</code>
62	B	<code>print("Hello","World",sep='-',end='!')='Hello-World!'</code>
63	B	property no setter => error on assign after <code>print(5)</code>
64	C	<code>range(1,4)=1,2,3</code>
65	C	no <code>repr</code> => < main .Point object at 0x...>
66	B	set removes duplicates => {1,2,3}
67	B	<code>setattr(b,'title','Python Programming') => b.title='Python Programming'</code>
68	A	<code>text[7:12]='World'</code>
69	B	sorted by len => ['apple','banana','cherry']
70	B	sorted dict keys => ['a','b','c']
71	A	staticmethod add(3,4)=7
72	A	<code>str(100)='100', "Number is: 100"</code>
73	C	<code>sum([1,2,3],10)=16</code>
74	A	super greet => "Hello from Parent" then "Hello from Child"
75	A	super greet(name) => "Hello, Diana from Parent" "Hello, Diana from Child"
76	B	<code>tuple([1,2,3])=(1,2,3)</code>
77	B	<code>type(3.14)=float,type("Hello")=str</code>
78	A	<code>vars()</code> in func {'a':1,'b':2}
79	B	<code>vars()</code> global includes name ,x,y,...
80	B	zip stops shortest => [('Alice',25),('Bob',30)]

Licensed for: WBS Training AG
PYDASC71e-13



3.1 - Section 1: Modules and Packages

This section (Max. **12%** or 6x questions of the PCAP exam) focuses on how Python organizes and manages code at scale. You'll learn about modules, packages, and how Python's import system works. The questions will touch on topics like `__init__.py`, `sys.path`, the `import` statement, `__all__`, and best practices for structuring your code into reusable components. After this quiz, you'll be more confident in organizing your projects into cleanly separated modules and packages.

Quiz

It contains 50x questions. Please answer the questions on separate paper or in an Excel file, and then compare your answers with those in the answers section after the quiz.

Q1 (Modules & Packages):

What does `import math` do?

- It imports the math module and makes it available as `math` in the current namespace.
- It imports only the `ceil` and `floor` functions from math.
- It automatically imports all installed third-party math libraries.
- It clears `sys.path` before importing.

Q2 (Modules & Packages):

What is the purpose of `__init__.py` in a Python package?

- It indicates that the directory is a package, allowing imports from it.
- It stores the bytecode for the package.
- It prevents the package from being imported.
- It always executes before any module in the system.

Q3 (Modules & Packages):

What does the `dir()` function do when called on a module?

- It returns a list of the attributes and names defined in that module.
- It returns the file path of the module.
- It returns the docstring of the module.
- It lists only the built-in functions of Python.

Q4 (Modules & Packages):

What is `sys.path`?

- A list of directories that the interpreter searches for modules to import.
- A single string containing the current working directory.

- A dictionary mapping module names to file paths.
- An environment variable containing OS paths.

❓ Q5 (Modules & Packages):

Given:

```
import math
print(math.ceil(3.7))
```

What is printed?

- 4
- 3
- 3.7
- Error, because `ceil` is not defined in math.

❓ Q6 (Modules & Packages):

What is the effect of running:

```
import sys
sys.path.append('/custom/path')
```

- `/custom/path` is added to the module search path, allowing imports from that directory.
- It removes all other directories from `sys.path`.
- It raises a `SyntaxError`.
- It sets the current working directory to `/custom/path`.

❓ Q7 (Modules & Packages):

What does `from math import floor` do?

- It imports only `floor` from the math module into the current namespace.
- It imports the entire math module as `floor`.
- It imports nothing unless you call `floor()` immediately.
- It clears the current namespace.

❓ Q8 (Modules & Packages):

What happens if you try to import a module that doesn't exist?

- A `ModuleNotFoundError` (or `ImportError` in older versions) is raised.
- Python silently ignores it.
- It returns `None` and continues.
- It creates an empty module placeholder.

❓ Q9 (Modules & Packages):

What is `__pycache__`?

- A directory where compiled bytecode files (`.pyc`) are stored to speed up imports.
- A variable that holds cached versions of functions.
- A file containing all module names.
- A built-in function that clears memory.

❓ Q10 (Modules & Packages):

What is `__name__` in a Python module?

- A special variable that holds the name of the module. If the module is run as the main script, `__name__` is "`__main__`".
- Always the filename of the module without extension.
- Always "module".
- A variable that is always empty.

❓ Q11 (Modules & Packages):

What does `from random import choice` do?

- It imports the `choice` function from the `random` module into the current namespace.
- It imports the entire `random` module as `choice`.
- It imports nothing unless `choice` is called.
- It removes all functions except `choice` from `random`.

❓ Q12 (Modules & Packages):

If we have `__all__ = ['x', 'y']` in a module, what does `from module import *` do?

- It imports only the names `x` and `y`.
- It imports every name defined in the module.
- It raises an error if any other names exist.
- It imports nothing.

❓ Q13 (Modules & Packages):

What does `import math as m` do?

- It imports the `math` module but makes it accessible as `m` instead of `math`.
- It imports only functions that start with 'm'.
- It creates an alias for `math` but doesn't actually import it.
- It overwrites built-in functions.

❓ Q14 (Modules & Packages):

Consider:

```
from math import *
print(floor(4.5))
```

Is this valid?

- Yes, it imports all public names from math, including `floor`.
- No, `import *` is not allowed in Python.
- It imports nothing, `floor` is undefined.
- It raises a `SyntaxError`.

❓ Q15 (Modules & Packages):

How do you inspect what can be imported from a module?

- Use `dir(module)` to list attributes and functions.
- Use `print(module.__source__)`.
- Use `help()` always returns only built-in functions.
- It's not possible; modules are opaque.

❓ Q16 (Modules & Packages):

What does `import platform` allow?

- It allows using `platform` functions like `platform.system()`, `platform.version()`.
- It installs a new system platform.
- It only imports OS-specific modules.
- It is a syntax error.

❓ Q17 (Modules & Packages):

Why might you use `random.seed()`?

- To make the sequence of random numbers reproducible.
- To remove randomness.
- To import only deterministic functions from random.
- It's required before calling any random function.

❓ Q18 (Modules & Packages):

What does `random.choice([1,2,3])` return?

- A randomly selected element from the list `[1,2,3]`.
- Always 1.
- The entire list.
- An empty list.

❓ Q19 (Modules & Packages):

How to discover the current search paths for importing modules?

- Check `sys.path`.
- Print `os.environ['PATH']`.
- Use `dir(sys)`.
- It's hard-coded and cannot be viewed.

❓ Q20 (Modules & Packages):

What is `from package import module`?

- It imports `module` from within `package`, assuming `package` is a directory with `__init__.py`.
- It imports the entire Python environment.
- It imports `package` from `module`.
- It's not valid syntax.

❓ Q21 (Modules & Packages):

What does `import mypkg.subpkg.mod` imply?

- That `mypkg` is a package, `subpkg` is a sub-package, and `mod` is a module inside `subpkg`.
- That `mypkg` is a module containing `subpkg`.
- That `mod` is a built-in Python module.
- It imports nothing and raises `ImportError`.

❓ Q22 (Modules & Packages):

If `mypkg/__init__.py` is empty, what does this mean?

- `mypkg` is recognized as a package directory, but no special initialization is performed.
- The package cannot be imported.
- It will import all modules in that directory by default.
- It deletes `mypkg` after loading.

❓ Q23 (Modules & Packages):

What is a Python package?

- A directory with `__init__.py` that can contain modules or sub-packages.
- A single `.py` file is a package.
- A C library that Python can call.
- A zip file of source code.

❓ Q24 (Modules & Packages):

What is the difference between a package and a module?

- A package is a directory with `__init__.py`, a module is a single `.py` file.
- They are the same.
- A module can contain packages, but not vice versa.

- A package is always written in C, a module in Python.

❓ Q25 (Modules & Packages):

What does `math.floor(-3.7)` return?

- -4
- -3
- 3
- 4

❓ Q26 (Modules & Packages):

What does `platform.system()` typically return?

- The name of the operating system (e.g., "Windows", "Linux", "Darwin").
- The Python version.
- The CPU architecture.
- The hostname of the machine.

❓ Q27 (Modules & Packages):

What does `platform.machine()` return?

- A string identifying the machine hardware, e.g. 'x86_64'.
- The OS name.
- The Python interpreter name.
- A list of all installed packages.

❓ Q28 (Modules & Packages):

What does `platform.python_version()` return?

- A string with the Python version number, e.g. '3.10.0'.
- The OS version.
- The path to the Python executable.
- The build time of Python.

❓ Q29 (Modules & Packages):

What is `__package__` attribute in a module?

- It indicates the package name that the module belongs to.
- It stores the version of the package.
- It's always empty.
- It replaces `__name__`.

❓ Q30 (Modules & Packages):

What happens when a module is imported multiple times?

- The module is only executed once and then cached; subsequent imports fetch it from `sys.modules`.
- It executes the module code each time.
- It raises a `RuntimeError`.
- It deletes previously imported modules.

❓ Q31 (Modules & Packages):

What does `sys.modules` represent?

- A dictionary mapping module names to module objects already loaded.
- A list of available modules on the system.
- An environment variable for modules.
- It's deprecated and not used.

❓ Q32 (Modules & Packages):

If you have `from mypkg import *` and `__all__ = ['mod1']` in `mypkg/__init__.py`, what is imported?

- Only `mod1`.
- Everything in `mypkg`.
- Nothing, since `*` is not allowed.
- Raises `ImportError`.

❓ Q33 (Modules & Packages):

What is the effect of `import mypkg.module as m`?

- Imports `mypkg.module` and assigns it the local name `m`.
- Renames `mypkg` to `m`.
- Imports all modules inside `mypkg` that start with `m`.
- Does nothing unless `m` is defined.

❓ Q34 (Modules & Packages):

What does `from mypkg.subpkg import mod` assume?

- That `mypkg` and `subpkg` are packages (directories with `__init__.py`), and `mod` is a module.
- That `mod` is a directory.
- That `subpkg` is a single Python file.
- That `mypkg` is on `sys.path`.

❓ Q35 (Modules & Packages):

What is the recommended way to organize user-defined modules?

- Put them in separate `.py` files and if grouping them, use packages.
- All code in a single massive `.py` file.
- Store them in system directories without `__init__.py`.
- They must be compiled into `.pyc` before use.

?

Q36 (Modules & Packages):

Why might you use `import random; random.seed(42)` in testing?

- To ensure the same random results every run, aiding reproducible tests.
- To disable randomness altogether.
- To speed up the random module.
- To import all standard library modules.

?

Q37 (Modules & Packages):

What does `help(math)` do in a Python shell?

- Displays the documentation for the math module.
- Returns a list of math functions as a list.
- Installs math related packages.
- Clears the screen.

?

Q38 (Modules & Packages):

If `mypkg` is a package, can it contain another package `mypkg.subpkg`?

- Yes, Python supports nested packages.
- No, packages cannot be nested.
- Only if `subpkg` has no `__init__.py`.
- Only if `mypkg` is empty.

?

Q39 (Modules & Packages):

How is `if __name__ == "__main__":` used in modules?

- To check if the module is being run as a script, and if so, run some test code.
- To prevent the module from running.
- To rename the module.
- It's a deprecated syntax.

?

Q40 (Modules & Packages):

What does `from math import ceil, floor` do?

- Imports only `ceil` and `floor` from math.
- Imports the entire math module twice.
- Errors if both `ceil` and `floor` are imported at once.
- Imports nothing.

?

Q41 (Modules & Packages):

If you run `print(__name__)` in a module file executed directly, what is printed?

- `__main__`

- The module's filename
- An empty string
- A random GUID

❓ Q42 (Modules & Packages):

What is the main purpose of `platform` module?

- To provide information about the underlying platform (OS, version, machine).
- To manage Python packages.
- To handle math functions.
- To randomize output.

❓ Q43 (Modules & Packages):

How do you list attributes of `random` module?

- `dir(random)`
- `random.__all__` always lists everything.
- `help(dir).`
- You cannot list them.

❓ Q44 (Modules & Packages):

What happens if `__init__.py` in a package sets `__all__ = []`?

- `from package import *` imports nothing from that package.
- It imports all modules anyway.
- It raises `ImportError`.
- It deletes the package.

❓ Q45 (Modules & Packages):

If `mypkg/__init__.py` has `from . import mod`, what does it do?

- Imports `mod` from the same package `mypkg`.
- Imports from the parent directory.
- Imports a system-level module named `mod`.
- Errors unless `mod` is a built-in name.

❓ Q46 (Modules & Packages):

What is a "public" variable in a module?

- A variable not starting with underscore, by convention accessible as a public API.
- A variable declared with `public` keyword.
- A variable defined in `__all__` only.
- Python doesn't have public variables concept at all; only naming conventions.

❓ Q47 (Modules & Packages):

If `mod.py` contains `_secret = 42`, what does `_secret` represent?

- By convention, it's a "private" variable not intended for public use.
- It's enforced by Python to be inaccessible.
- It is a built-in variable.
- It raises a syntax error.

❓ Q48 (Modules & Packages):

What does `math.sqrt(16)` return?

- 4.0
- 4
- 16
- Error, sqrt not in math.

❓ Q49 (Modules & Packages):

If you do `import math` and then `import math` again, how many times is math's code executed?

- Once; subsequent imports reuse the cached module.
- Twice; each import runs it again.
- Zero times; math is built-in.
- It depends on the system.

❓ Q50 (Modules & Packages):

If `mypkg` defines `__all__ = ['mod1']` in `mypkg/_init__.py`, and you do `from mypkg import *`, what is accessible?

- Only `mod1` is accessible directly.
- All submodules of `mypkg`.
- Nothing, `__all__` is ignored.
- It raises `ImportError`.

✓ The Answers

Q#	Answer	Explanation
1	A	<code>import math</code> imports the math module into current namespace as <code>math</code> .
2	A	<code>__init__.py</code> file marks a directory as a package.
3	A	<code>dir(module)</code> lists module attributes.
4	A	<code>sys.path</code> is the search list for modules.
5	A	<code>math.ceil(3.7)=4</code> .
6	A	<code>sys.path.append</code> adds a new directory for module searches.
7	A	<code>from math import floor</code> imports only <code>floor</code> .
8	A	Attempting to import a non-existing module raises <code>ModuleNotFoundError</code> .
9	A	<code>__pycache__</code> stores compiled bytecode.
10	A	<code>__name__</code> is " <code>main</code> " if run directly, else module name.
11	A	<code>from random import choice</code> imports only <code>choice</code> .
12	A	<code>__all__</code> controls what <code>from ... import *</code> imports.
13	A	<code>import math as m</code> gives an alias <code>m</code> .
14	A	<code>from math import *</code> imports public names including <code>floor</code> .
15	A	<code>dir(module)</code> can inspect module contents.
16	A	<code>import platform</code> allows <code>platform.*</code> functions.
17	A	<code>random.seed()</code> ensures reproducible random sequences.
18	A	<code>random.choice</code> selects a random element.
19	A	<code>sys.path</code> shows import search paths.
20	A	<code>from package import module</code> imports module from package.
21	A	<code>mypkg.subpkg.mod</code> implies nested structure.
22	A	Empty <code>init.py</code> still defines a package.
23	A	A package is a directory with <code>init.py</code> .
24	A	Module=single file, Package=dir with <code>init.py</code> .
25	A	<code>floor(-3.7)=-4</code> .
26	A	<code>platform.system()</code> returns OS name.
27	A	<code>platform.machine()</code> gives machine type.
28	A	<code>platform.python_version()</code> returns Python version string.
29	A	package is the package name a module is in.

Q#	Answer	Explanation
30	A	Modules are cached after first import.
31	A	sys.modules is a dict of loaded modules.
32	A	from package import * with all imports listed names.
33	A	import mypkg.module as m gives alias m.
34	A	from mypkg.subpkg import mod means nested packages.
35	A	Separate .py files and packages is recommended.
36	A	Seeding random for tests ensures reproducibility.
37	A	help(math) shows math docs.
38	A	Nested packages are supported.
39	A	if name=="main" checks if run as script.
40	A	from math import ceil,floor imports those two.
41	A	Running module directly sets name="main" .
42	A	platform module gives platform/OS info.
43	A	dir(random) lists attributes.
44	A	all=[] imports nothing with *.
45	A	from . import mod imports mod from same package.
46	A	Public by convention means no leading underscore.
47	A	Leading underscore means "private" by convention.
48	A	math.sqrt(16)=4.0.
49	A	A module is executed once and cached.
50	A	all in init.py restricts what * imports.



3.2 - Section 2: Exceptions

In this section (Max. **14%** or 5x questions of the PCAP exam), you'll be exploring Python's error and exception handling mechanisms. The questions will cover concepts such as `try/except` blocks, `finally` clauses, custom exceptions, and the behavior of built-in exception types. By the end of this quiz, you'll have a clearer understanding of how Python deals with runtime errors and how you can write more robust, error-resilient code.

Quiz

It contains 50x questions. Please answer the questions on separate paper or in an Excel file, and then compare your answers with those in the answers section after the quiz.

Q1 (Exceptions):

What happens if a `ZeroDivisionError` occurs and there is an appropriate `except` block for it?

```
try:  
    x = 1/0  
except ZeroDivisionError:  
    print("Cannot divide by zero!")
```

- It prints "Cannot divide by zero!"
- It prints nothing
- It raises `ZeroDivisionError` without handling
- It executes the `else` block if any

Q2 (Exceptions):

What does the following code print?

```
try:  
    x = int("abc")  
except ValueError:  
    print("Value error encountered!")  
else:  
    print("No error!")
```

- "Value error encountered!"
- "No error!"
- Nothing
- Error

❓ Q3 (Exceptions):

What does the `finally` block do in a try-except-finally structure?

- It always executes after try and except blocks, regardless of whether an exception occurred.
- It executes only if no exceptions occurred.
- It never executes if an exception is raised.
- It executes before the except block.

❓ Q4 (Exceptions):

What does the code print?

```
try:  
    x = int("10")  
except ValueError:  
    print("Value error")  
else:  
    print("Converted successfully!")  
finally:  
    print("Done")
```

- "Converted successfully!" followed by "Done"
- "Value error" followed by "Done"
- "Done" only
- Nothing

❓ Q5 (Exceptions):

What does `raise` do?

- It explicitly raises an exception.
- It logs an error message without stopping execution.
- It ignores exceptions.
- It retries the block of code.

❓ Q6 (Exceptions):

What type of exception is raised by `int("abc")`?

- ValueError
- TypeError
- ZeroDivisionError
- KeyError

❓ Q7 (Exceptions):

What happens if no except block matches the raised exception?

- The exception propagates up and can terminate the program if not handled.
- It prints "Unhandled exception" and continues.
- It silently ignores the error.
- It converts the exception to ValueError.

❓ Q8 (Exceptions):

Can you have multiple except blocks for different exceptions?

```
try:  
    # code  
except ValueError:  
    # handle ValueError  
except TypeError:  
    # handle TypeError
```

- Yes, Python checks them in order and uses the first matching one.
- No, only one except block allowed.
- It raises SyntaxError.
- They must be combined into one except.

❓ Q9 (Exceptions):

What does `assert condition, "message"` do if condition is False?

- Raises an AssertionError with "message"
- Prints "message" and continues
- Does nothing
- Raises ValueError

❓ Q10 (Exceptions):

What is the purpose of a custom exception class?

- To define user-specific error types for clearer error handling.
- To replace built-in exceptions entirely.
- To make code run faster.
- They are not allowed in Python.

❓ Q11 (Exceptions):

What will this code print?

```
try:  
    x = 10 / 0  
except ZeroDivisionError as e:  
    print("Error:", e)
```

- "Error: division by zero"
- "Error:"
- Nothing
- Raises ZeroDivisionError unhandled.

❓ Q12 (Exceptions):

What is the effect of `except Exception:`?

- It catches any exception derived from Exception.
- It only catches system exit signals.
- It only catches ZeroDivisionError.
- It is invalid syntax.

❓ Q13 (Exceptions):

Is `finally` block executed if an exception is raised and not caught?

```
try:  
    1/0  
finally:  
    print("Cleanup")
```

- Yes, "Cleanup" is printed before the exception propagates.
- No, it never executes.
- Only if we have an except block.
- Raises another error.

❓ Q14 (Exceptions):

What does `raise` without arguments do inside an except block?

- It re-raises the current exception.
- It raises a new ValueError.
- It does nothing.
- It clears the exception.

❓ Q15 (Exceptions):

What exception is raised by `assert False, "fail"`?

- AssertionError with message "fail"
- ValueError
- TypeError
- No exception, it prints "fail"

❓ Q16 (Exceptions):

Can the else block after try-except run if an exception occurred?

- No, else executes only if no exceptions were raised in the try block.
- Yes, it always executes.
- It executes before except.
- It executes only if there's a finally block.

❓ Q17 (Exceptions):

What does this print?

```
try:  
    x = float("3.14")  
except ValueError:  
    print("Value error")  
else:  
    print("No error")  
finally:  
    print("Done")
```

- "No error" then "Done"
- "Value error" then "Done"
- "Done" only
- Error

❓ Q18 (Exceptions):

What is the **try-except-else-finally** order of execution?

- try -> if exception caught: except -> else (only if no exception) -> finally always last
- try -> else -> except -> finally
- except -> try -> else -> finally
- try -> finally -> except -> else

❓ Q19 (Exceptions):

What exception is raised by **int("3.14")**?

- ValueError
- TypeError
- ZeroDivisionError
- KeyError

❓ Q20 (Exceptions):

Which exception is raised when a nonexistent dictionary key is accessed?

- KeyError
- ValueError

- IndexError
- TypeError

❓ Q21 (Exceptions):

What does this code print?

```
try:  
    d = {"a":1}  
    print(d["b"])  
except KeyError as e:  
    print("Missing key:", e)
```

- "Missing key: 'b'"
- "Missing key:"
- Nothing
- KeyError unhandled

❓ Q22 (Exceptions):

What happens if `assert True` is executed?

- Nothing, it passes quietly.
- It raises AssertionError.
- It prints True.
- It returns True.

❓ Q23 (Exceptions):

When defining a custom exception, which class should it typically inherit from?

- Exception (or a subclass of it)
- int
- object only
- No inheritance needed

❓ Q24 (Exceptions):

How do you raise a custom exception?

```
class MyError(Exception):  
    pass  
  
# ...
```

- `raise MyError("message")`
- `MyError("message")`

- except MyError:
- assert MyError

❓ Q25 (Exceptions):

What does `except (ValueError, TypeError):` do?

- Catches both ValueError and TypeError exceptions.
- Catches all exceptions.
- Catches no exceptions.
- Raises SyntaxError.

❓ Q26 (Exceptions):

If multiple except blocks match, which is chosen?

- The first matching except block from top to bottom.
- The last matching except block.
- Randomly chosen.
- Raises SyntaxError.

❓ Q27 (Exceptions):

What exception is raised by accessing a list index out of range?

- IndexError
- KeyError
- ValueError
- TypeError

❓ Q28 (Exceptions):

What does this code print?

```
try:  
    raise IOError("I/O error occurred")  
except IOError as e:  
    print(e)
```

- "I/O error occurred"
- "IOError"
- Nothing
- Unhandled exception

❓ Q29 (Exceptions):

Is `except:` with no exception type allowed?

- Yes, it catches any exception (not recommended).

- No, it's invalid syntax.
- Only in Python 2.
- It only catches syntax errors.

❓ Q30 (Exceptions):

What block always runs after try and except?

- finally block always runs.
- else block always runs.
- except block always runs.
- None.

❓ Q31 (Exceptions):

What happens here?

```
try:  
    pass  
except:  
    print("Error")  
else:  
    print("No error")  
finally:  
    print("Always")
```

- "No error" then "Always"
- "Error" then "Always"
- "Always" only
- Nothing

❓ Q32 (Exceptions):

Which exception is raised by `assert 1==2`?

- AssertionError
- ValueError
- TypeError
- IndexError

❓ Q33 (Exceptions):

What is printed?

```
try:  
    x = 1/0  
except ZeroDivisionError:  
    print("ZeroDiv")
```

```
except Exception:  
    print("General")
```

- "ZeroDiv"
- "General"
- Both
- Nothing

❓ Q34 (Exceptions):

What is the purpose of `raise` inside an except block?

```
try:  
    1/0  
except ZeroDivisionError:  
    raise
```

- It re-raises the ZeroDivisionError after catching it.
- It ignores the error.
- It raises a new ValueError.
- It prints nothing.

❓ Q35 (Exceptions):

What exception does `open("nonexistent.txt", "r")` raise if the file does not exist?

- FileNotFoundError
- ValueError
- TypeError
- KeyError

❓ Q36 (Exceptions):

What does `try: ... except Exception as e:` do?

- It catches any exception derived from Exception and assigns it to `e`.
- Catches only NameError.
- Catches no exceptions.
- It's invalid syntax.

❓ Q37 (Exceptions):

What exception is raised by `int("abc")`?

- ValueError
- TypeError
- KeyError

- IndexError

❓ Q38 (Exceptions):

What does else block do in try-except-else?

- Executes only if the try block did not raise any exception.
- Executes only if an exception is raised.
- Always executes before except.
- It never executes.

❓ Q39 (Exceptions):

What happens if `assert` is disabled with `-O` flag at runtime?

- Assert statements are skipped and don't raise errors.
- They raise `AssertionError` anyway.
- They print a warning.
- They become syntax errors.

❓ Q40 (Exceptions):

Can a custom exception carry additional data?

- Yes, by defining `init` and storing attributes.
- No, exceptions are fixed.
- Only if inherited from `KeyError`.
- Only built-in exceptions can carry data.

❓ Q41 (Exceptions):

What does this code print?

```
class MyError(Exception):
    pass

try:
    raise MyError("Failed")
except MyError as e:
    print("Caught:", e)
```

- "Caught: Failed"
- "Caught: MyError"
- "Caught:"
- Nothing

❓ Q42 (Exceptions):

What is the difference between `except ValueError:` and `except ValueError as e:`?

- `as e` assigns the exception instance to `e`, without '`as e`' doesn't.
- No difference.
- `as e` catches no exceptions.
- It changes the exception type.

❓ Q43 (Exceptions):

What does `raise ValueError("bad") from None` do?

- Raises `ValueError("bad")` without chaining a previous exception.
- Raises `NoneType` error.
- It ignores `ValueError`.
- `SyntaxError`.

❓ Q44 (Exceptions):

What does `except (ValueError, TypeError) as e:` mean?

- Catches either `ValueError` or `TypeError` and assigns the exception to `e`.
- Catches no exceptions.
- `Syntax error`.
- Only catches `ValueError`.

❓ Q45 (Exceptions):

Can you use finally without except?

```
try:  
    # code  
finally:  
    # code
```

- Yes, try-finally is valid and runs finally regardless of exceptions.
- No, must have `except`.
- Raises `SyntaxError`.
- Ignores finally.

❓ Q46 (Exceptions):

What is printed?

```
try:  
    a = [1,2]  
    print(a[5])  
except IndexError:  
    print("Out of range")
```

- "Out of range"
- Nothing
- Raises IndexError
- "Out of range" then error

❓ Q47 (Exceptions):

What exception is raised if `import nonexistent_module`?

- ModuleNotFoundError
- ImportError in older Python versions
- ValueError
- KeyError

❓ Q48 (Exceptions):

Does the else block run if an exception occurs?

- No, else runs only if no exception occurred in try.
- Yes, always.
- Only if finally is present.
- It depends on the exception.

❓ Q49 (Exceptions):

What does `assert x >= 0, "x must be nonnegative"` do if x is -1?

- Raises AssertionError
- Prints "x must be nonnegative"
- Returns False
- Does nothing

❓ Q50 (Exceptions):

What happens if `raise` is used outside an except block without arguments?

- It raises a RuntimeError as there's no active exception to re-raise.
- It does nothing.
- It raises SyntaxError.
- It prints a warning.

✓ The Answers

Q#	Answer	Explanation
1	A	except ZeroDivisionError handles the division by zero
2	A	int("abc") raises ValueError, caught by except block
3	A	finally always runs regardless of exceptions
4	A	int("10") no error => else, then finally
5	A	raise explicitly throws an exception
6	A	int("abc") -> ValueError
7	A	Uncaught exception propagates and may terminate program
8	A	Multiple except allowed, checked in order
9	A	assert fails => AssertionError with message
10	A	Custom exceptions create clear error types
11	A	division by zero => ZeroDivisionError's message printed
12	A	except Exception catches all standard exceptions
13	A	finally runs even if exception not caught
14	A	raise inside except re-raises current exception
15	A	assert False raises AssertionError
16	A	else runs only if no exception in try
17	A	float("3.14") no error => else then finally
18	A	else after except runs only if no exception in try
19	A	int("3.14") => ValueError
20	A	dict key not found => KeyError
21	A	d["b"] not found => KeyError caught, prints key
22	A	assert True does nothing
23	A	Custom exceptions typically inherit from Exception
24	A	raise MyError("message") throws custom exception
25	A	except (ValueError, TypeError) catches both
26	A	First matching except is chosen
27	A	Out-of-range index => IndexError
28	A	raise IOError("I/O error") caught prints message
29	A	except: catches all exceptions (not recommended)

Q#	Answer	Explanation
30	A	finally always runs
31	A	No error => else runs => "No error", then finally => "Always"
32	A	assert 1==2 fails => AssertionError
33	A	ZeroDivisionError caught by that except block
34	A	raise inside except re-raises the caught exception
35	A	FileNotFoundException if file not found
36	A	except Exception as e catches any standard exception
37	A	int("abc") => ValueError again
38	A	else runs if no exception in try
39	A	With -O asserts removed, no AssertionError raised
40	A	Custom exceptions can have custom attributes
41	A	Prints "Caught: Failed" from MyError
42	A	"as e" lets you access the exception instance
43	A	from None removes chaining info
44	A	Catches either ValueError or TypeError into e
45	A	try-finally is valid without except
46	A	a[5] out-of-range => IndexError caught => "Out of range"
47	A	nonexistent_module => ModuleNotFoundError
48	A	else doesn't run if exception occurred
49	A	x=-1 fails assert => AssertionError
50	A	raise without active exception => RuntimeError



3.3 - Section 3: Strings

This section (Max. **18%** or 8x questions of the PCAP exam) immerses you in one of Python's most fundamental data types: strings. Key topics include:

- String creation and immutability
 - Indexing and slicing strings
 - Common string methods (e.g., `.upper()`, `.lower()`, `.find()`, `.replace()`)
 - String formatting techniques
 - Working with escape characters and raw strings
 - Understanding and utilizing string comprehensions
-

Quiz

It contains 50x questions. Please answer the questions on separate paper or in an Excel file, and then compare your answers with those in the answers section after the quiz.

❓ Q1 (Strings):

What does the following code print?

```
s = "Hello"  
print(s[1])
```

- 'e'
- 'H'
- 'I'
- 'o'

❓ Q2 (Strings):

What is the result of the slicing?

```
s = "Python"  
print(s[2:5])
```

- 'tho'
- 'yth'
- 'on'
- 'Py'

❓ Q3 (Strings):

What does this print?

```
s = "Hello"  
print(s[-1])
```

- 'o'
- 'H'
- 'e'
- IndexError

❓ Q4 (Strings):

Which of the following methods returns the length of a string `s`?

- `len(s)`
- `s.length()`
- `count(s)`
- `size(s)`

❓ Q5 (Strings):

What does `s = "Hello" + "World"` produce?

- "HelloWorld"
- "Hello World"
- "Hello"
- "World"

❓ Q6 (Strings):

What does `"abc" * 3` produce?

- "abcabcabc"
- "abc3"
- "abcabc"
- Error

❓ Q7 (Strings):

Is a Python string mutable?

- No, strings in Python are immutable.
- Yes, you can change individual characters.
- It depends on Python version.
- Only if it's ASCII.

❓ Q8 (Strings):

What does `ord('A')` return?

- 65
- 'A'
- "65"
- 97

❓ Q9 (Strings):

What does `chr(97)` return?

- 'a'
- 97
- '97'
- 'A'

❓ Q10 (Strings):

Given `s = "Hello"`, what does `s.upper()` return?

- "HELLO"
- "hello"
- "Hello"
- "HeLLo"

❓ Q11 (Strings):

What does `"Hello".lower()` return?

- "hello"
- "HELLO"
- "HeLLo"
- "Hello"

❓ Q12 (Strings):

What does `"12345".isdigit()` return?

- True
- False
- Raises ValueError
- None

❓ Q13 (Strings):

What does `"Hello".isalnum()` return?

- True (all letters are alphanumeric)
- False
- Raises TypeError

- None

❓ Q14 (Strings):

What does "Hello World".split() return?

- ["Hello", "World"]
- "Hello World"
- ["Hello World"]
- ("Hello", "World")

❓ Q15 (Strings):

What does ",".join(["a", "b", "c"]) return?

- "a,b,c"
- "abc"
- ["a,b,c"]
- "a;b;c"

❓ Q16 (Strings):

If s = "banana", what does s.find("na") return?

- 2
- -1
- 1
- 3

❓ Q17 (Strings):

What does "banana".rfind("na") return?

- 4
- 2
- 3
- -1

❓ Q18 (Strings):

What does "hello".startswith("he") return?

- True
- False
- "he"
- Raises AttributeError

❓ Q19 (Strings):

What does "hello".endswith("lo") return?

- True
- False
- "lo"
- Raises AttributeError

❓ Q20 (Strings):

What does `"hello world".count("l")` return?

- 3
- 2
- 1
- 4

❓ Q21 (Strings):

Given `s = " spaced "`, what does `s.strip()` return?

- "spaced"
- " spaced "
- " spaced"
- "spaced "

❓ Q22 (Strings):

What does `"Hello".replace("l", "x")` return?

- "Hexxo"
- "Hello"
- "Hexo"
- "Heo"

❓ Q23 (Strings):

What is printed?

```
s = "abc"  
print("a" in s)
```

- True
- False
- 'a'
- Error

❓ Q24 (Strings):

What does `"hello"[-3:]` return?

- "llo"
- "hel"
- "lo"
- "hello"

❓ Q25 (Strings):

What does "`abcdef`[::2]" return?

- "ace"
- "abc"
- "adf"
- "af"

❓ Q26 (Strings):

What does "`Hello World`.find("X")" return?

- -1
- 0
- 5
- Error

❓ Q27 (Strings):

What does "`Hello\nWorld`" contain?

- A newline character between "Hello" and "World".
- A space instead of newline.
- Only ASCII letters.
- It's invalid.

❓ Q28 (Strings):

What does "`hello`".isalpha()" return?

- True
- False
- Error
- None

❓ Q29 (Strings):

What does " ".isspace()" return?

- True
- False
- Error
- None

❓ Q30 (Strings):

What does "Python3.12".isalnum() return?

- False
- True
- Error
- None

❓ Q31 (Strings):

What does len("Hello") return?

- 5
- 4
- 6
- "5"

❓ Q32 (Strings):

What does "abc" < "abd" return?

- True
- False
- Error
- None

❓ Q33 (Strings):

What does "abc" > "aba" return?

- True
- False
- Error
- None

❓ Q34 (Strings):

What does "abc" == "abc" return?

- True
- False
- Error
- None

❓ Q35 (Strings):

What does "abc" * 0 return?

- "" (empty string)

- "abc"
- Error
- None

❓ Q36 (Strings):

What does `"Hello".find("l", 3)` do?

- Finds 'l'
- Returns 2
- Returns -1
- Error

❓ Q37 (Strings):

What does `"hello".index("ll")` return?

- 2
- -1
- 1
- 3

❓ Q38 (Strings):

What does `"Banana".capitalize()` return?

- "Banana"
- "BANANA"
- "banana"
- "BanAna"

❓ Q39 (Strings):

What does `"HELLO".lower()` return?

- "hello"
- "HELLO"
- "Hello"
- "heLLo"

❓ Q40 (Strings):

What does `"hello".title()` return?

- "Hello"
- "hello"
- "Hello World"
- "HELLO"

❓ Q41 (Strings):

What does `"hello world".split("o")` return?

- ['hell', ' w', 'rld']
- ["hello world"]
- ['hell', 'world']
- ['h','e','l','l','o',' ','w','o','r','l','d']

❓ Q42 (Strings):

What is `"".join(["H", "i"])`?

- "Hi"
- "H i"
- ['H', 'i']
- ""

❓ Q43 (Strings):

What does `"Hello".center(10)` return?

- " Hello "
- "Hello"
- "Hello "
- " Hello"

❓ Q44 (Strings):

What does `"banana".find("z")` return?

- -1
- 0
- 2
- Raises ValueError

❓ Q45 (Strings):

What does `"banana".replace("na", "NA")` return?

- "baNANA"
- "banaNA"
- "bANAna"
- "baNAna"

❓ Q46 (Strings):

What does `" test ".lstrip()` return?

- "test "
- "test"
- " test"

- " test "

❓ Q47 (Strings):

What does `"abc".encode()` do by default?

- Returns a bytes object (b'abc') encoded in UTF-8 by default.
- Returns the same string unchanged.
- Raises an error.
- Converts to uppercase.

❓ Q48 (Strings):

What does `"hello".find("l",4)` return?

- -1
- 2
- 3
- 4

❓ Q49 (Strings):

What does `"Hello\tWorld".expandtabs(4)` do?

- Replaces '\t' with spaces so that tab = 4 spaces, e.g. "Hello World"
- Removes the tab
- Replaces tab with '\n'
- Does nothing

❓ Q50 (Strings):

What does `"mississippi".count("iss")` return?

- 2
 - 1
 - 3
 - 4
-

✓ The Answers

Q#	Answer	Explanation
1	A	s[1]='e' in "Hello"
2	A	s[2:5]="tho" from "Python"
3	A	s[-1] = last char = 'o'
4	A	len(s) returns string length
5	A	String concatenation: "HelloWorld"
6	A	"abc"*3 = "abcabcabc"
7	A	Strings are immutable in Python
8	A	ord('A')=65
9	A	chr(97)='a'
10	A	upper() converts to uppercase: "HELLO"
11	A	lower() -> "hello"
12	A	"12345".isdigit()=True
13	A	"Hello".isalnum()=True (only letters are alnum)
14	A	split() with no args splits on whitespace -> ["Hello","World"]
15	A	",".join(["a","b","c"])="a,b,c"
16	A	"banana".find("na")=2
17	A	rfind("na") in "banana"=4
18	A	"hello".startswith("he")=True
19	A	endswith("lo")=True
20	A	count('l') in "hello world"=3
21	A	strip removes leading/trailing spaces => "spaced"
22	A	replace("l","x") in "Hello"="Hexxo"
23	A	'a' in "abc"=True
24	A	"hello"[-3:]="llo"
25	A	"abcdef"[::2]="ace"
26	A	find("X") not found => -1
27	A	"\n" is newline
28	A	"hello".isalpha()=True
29	A	" ".isspace()=True

Q#	Answer	Explanation
30	A	"Python3.12".isalnum()=False due to '.'
31	A	len("Hello")=5
32	A	"abc" < "abd"=True ('c'<'d')
33	A	"abc" > "aba"=True ('c'>'a')
34	A	"abc"=="abc"=True
35	A	"abc"**0=""
36	A	find('l',3) finds 'l' at index 3
37	A	"hello".index("ll")=2
38	A	"Banana".capitalize()="Banana"
39	A	"HELLO".lower()="hello"
40	A	"hello".title()="Hello"
41	A	split("o"): "hello world" -> ['hell','w','rld']
42	A	"".join(["H","i"])="Hi"
43	A	center(10) pads "Hello" with spaces => " Hello "
44	A	find("z")=-1 not found
45	A	replace("na","NA"): "banana"->"baNANA"
46	A	lstrip removes leading spaces => "test "
47	A	encode() returns b'abc' by default UTF-8
48	A	find("l",4): no 'l' after index4 => -1
49	A	expandtabs(4): '\t' replaced by 4 spaces
50	A	"mississippi".count("iss")=2



3.4 - Section 4: Object-Oriented Programming (OOP)

This section (Max. **34%** or 12x questions of the PCAP exam) investigates Python's object-oriented programming features. You'll review classes, objects, inheritance, polymorphism, encapsulation, and special methods like `__init__` and `__str__`. The questions will help solidify your understanding of how Python's OOP model works, allowing you to design more structured, reusable, and maintainable code.

Quiz

It contains 50x questions. Please answer the questions on separate paper or in an Excel file, and then compare your answers with those in the answers section after the quiz.

Q1 (OOP):

What does the following code print?

```
class MyClass:  
    x = 10  
    def __init__(self, y):  
        self.y = y  
  
obj = MyClass(20)  
print(obj.x, obj.y)
```

- 20 10
- Error
- None None
- 10 20

Q2 (OOP):

What is printed?

```
class A:  
    value = 5  
  
a = A()  
A.value = 10  
print(a.value)
```

- 5
- 10
- None

- Error

❓ Q3 (OOP):

Consider the code:

```
class A:  
    def __init__(self):  
        self.data = 42  
  
a = A()  
print(a.__dict__)
```

- {'data': 42}
- {}
- Error
- {'dict': {}}

❓ Q4 (OOP):

What does the following print?

```
class A:  
    def method(self):  
        return "A"  
  
class B(A):  
    pass  
  
b = B()  
print(b.method())
```

- A
- B
- Error
- None

❓ Q5 (OOP):

What is printed?

```
class A:  
    __hidden = 100  
  
a = A()  
print(a._A__hidden)
```

- 100
- Error
- __hidden
- None

❓ Q6 (OOP):

What does this code print?

```
class A:  
    def __init__(self, x):  
        self.x = x  
  
    def __str__(self):  
        return f"Value: {self.x}"  
  
a = A(10)  
print(a)
```

- Value: 10
- 10
- A object
- Error

❓ Q7 (OOP):

What is printed?

```
class Base:  
    def greet(self):  
        return "Hello from Base"  
  
class Derived(Base):  
    pass  
  
d = Derived()  
print(isinstance(d, Base), d.greet())
```

- True Hello from Base
- True Hello from Derived
- False Hello from Base
- Error

❓ Q8 (OOP):

What does this print?

```
class A:  
    pass  
  
class B(A):  
    pass  
  
b = B()  
print(isinstance(B, A))
```

- True
- False
- Error
- None

❓ Q9 (OOP):

What is the result?

```
class X:  
    val = 1  
  
x1 = X()  
x2 = X()  
x2.val = 2  
X.val = 3  
print(x1.val, x2.val)
```

- 3 2
- 1 2
- 3 3
- 2 3

❓ Q10 (OOP):

What is printed?

```
class A:  
    def __init__(self):  
        print("A init")  
  
class B(A):  
    def __init__(self):  
        super().__init__()  
        print("B init")  
  
b = B()
```

- A init\nB init
- B init\nA init
- A init only
- B init only

❓ Q11 (OOP):

What does this print?

```
class A:  
    def __init__(self,x):  
        self.x = x  
    def double(self):  
        return self.x*2  
  
a = A(5)  
print(a.double())
```

- 10
- 5
- Error
- None

❓ Q12 (OOP):

Given:

```
class A:  
    pass  
  
a = A()  
print(a.__dict__)
```

- {}

- {'dict':{}}
- Error
- None

❓ Q13 (OOP):

What is printed?

```
class A:  
    val = 10  
  
a = A()  
A.val = 20  
print(a.val)
```

- 20
- 10
- Error
- None

❓ Q14 (OOP):

Name mangling:

```
class A:  
    __secret = "hidden"  
  
a = A()  
print(hasattr(a, '_A__secret'))
```

- True
- False
- Error
- None

❓ Q15 (OOP):

Check inheritance:

```
class Base:  
    def greet(self):  
        return "Base"  
  
class Derived(Base):  
    def greet(self):  
        return "Derived"
```

```
d = Derived()  
print(d.greet())
```

- Derived
- Base
- Error
- None

❓ Q16 (OOP):

Class vs instance attributes:

```
class A:  
    val = 1  
  
a1 = A()  
a2 = A()  
a1.val = 2  
print(a1.val, a2.val, A.val)
```

- 2 1 1
- 1 1 1
- 2 2 2
- 2 1 2

❓ Q17 (OOP):

What does this print?

```
class A:  
    def __init__(self, x):  
        self.x = x  
  
a = A(10)  
print(a.x)
```

- 10
- None
- Error
- x

❓ Q18 (OOP):

What happens?

```
class A:  
    def __str__(self):  
        return "A instance"  
  
a = A()  
print(a)
```

- A instance
- <main.A object at ...>
- Error
- None

❓ Q19 (OOP):

Check class hierarchy:

```
class Super:  
    def hello(self):  
        return "Hello"  
  
class Sub(Super):  
    pass  
  
s = Sub()  
print(s.hello())
```

- Hello
- Error
- Sub object
- None

❓ Q20 (OOP):

Using super():

```
class A:  
    def greet(self):  
        return "A"  
  
class B(A):  
    def greet(self):  
        return super().greet() + "B"  
  
b = B()  
print(b.greet())
```

- AB
- A
- B
- Error

❓ Q21 (OOP):

Overriding **repr**:

```
class A:  
    def __init__(self,x):  
        self.x=x  
    def __repr__(self):  
        return f"A(x={self.x})"  
  
a=A(5)  
print(a)
```

- A(x=5)
- Value:5
- 5
- A object

❓ Q22 (OOP):

Check instance dictionary:

```
class A:  
    def __init__(self):  
        self.data=100  
  
a=A()  
print(a.__dict__)
```

- {'data':100}
- {}
- Error
- None

❓ Q23 (OOP):

Inheritance and `isinstance()`:

```
class Base: pass  
class Derived(Base): pass
```

```
d=Derived()  
print(isinstance(d,Base))
```

- True
- False
- Error
- None

❓ Q24 (OOP):

Multiple inheritance:

```
class A:  
    def method(self):  
        return "A"  
  
class B:  
    def method(self):  
        return "B"  
  
class C(A,B):  
    pass  
  
c = C()  
print(c.method())
```

- A
- B
- Error
- AB

❓ Q25 (OOP):

What is printed?

```
class A:  
    val = 5  
  
a = A()  
A.val = 10  
a.val = 20  
print(A.val, a.val)
```

- 10 20
- 5 10
- 5 20

- 10 10

❓ Q26 (OOP):

Constructor chaining:

```
class A:  
    def __init__(self):  
        self.data = "A"  
  
class B(A):  
    def __init__(self):  
        super().__init__()  
        self.data += "B"  
  
b=B()  
print(b.data)
```

- AB
- A
- B
- Error

❓ Q27 (OOP):

Polymorphism:

```
class Shape:  
    def area(self):  
        return 0  
  
class Square(Shape):  
    def __init__(self, side):  
        self.side=side  
    def area(self):  
        return self.side*self.side  
  
s = Square(3)  
print(s.area())
```

- 9
- 0
- Error
- None

❓ Q28 (OOP):

Name mangling with methods:

```
class A:  
    def __init__(self):  
        self.__val = 99  
    def __secret_method(self):  
        return self.__val  
  
a = A()  
print(a.__A__secret_method())
```

- 99
- Error
- __val
- None

❓ Q29 (OOP):

Check class attributes:

```
class A:  
    x=5  
  
class B(A):  
    pass  
  
b=B()  
print(b.x)
```

- 5
- Error
- None
- x

❓ Q30 (OOP):

Override init:

```
class A:  
    def __init__(self):  
        self.a="A"  
  
class B(A):  
    def __init__(self):  
        A.__init__(self)  
        self.b="B"  
  
b=B()  
print(b.a,b.b)
```

- A B
- B A
- Error
- None None

❓ Q31 (OOP):

Using isinstance():

```
class X: pass  
class Y(X): pass  
  
y=Y()  
print(isinstance(y,X))
```

- True
- False
- Error
- None

❓ Q32 (OOP):

Check if method belongs to superclass:

```
class Parent:  
    def greet(self):  
        return "Hi"  
  
class Child(Parent):  
    pass  
  
c=Child()  
print(c.greet())
```

- Hi
- Error
- Child
- None

❓ Q33 (OOP):

Default constructor:

```
class A:  
    pass  
  
a=A()  
print(hasattr(a, 'x'))
```

- False
- True
- Error
- None

❓ Q34 (OOP):

Method overriding:

```
class Animal:  
    def sound(self):  
        return "Generic sound"  
  
class Dog(Animal):  
    def sound(self):  
        return "Bark"  
  
d=Dog()  
print(d.sound())
```

- Bark
- Generic sound
- Error
- None

❓ Q35 (OOP):

Using **dict** on a class (not instance):

```
class A:  
    x=10
```

```
print(A.__dict__['x'])
```

- 10
- Error
- {}
- x

❓ Q36 (OOP):

Check class name:

```
class A:  
    pass  
  
print(A.__name__)
```

- A
- main
- Error
- " (empty string)

❓ Q37 (OOP):

Check module name:

```
class A:  
    pass  
  
print(A.__module__)
```

- main
- A
- "
- Error

❓ Q38 (OOP):

Overriding **repr** used by print if **str** not defined?

```
class A:  
    def __init__(self,x):  
        self.x=x  
    def __repr__(self):  
        return f"A({self.x})"
```

```
a=A(2)
print(a)
```

- A(2)
- 2
- A instance
- Error

❓ Q39 (OOP):

Inheritance chain:

```
class X:
    pass

class Y(X):
    pass

y=Y()
print(isinstance(y,Y), isinstance(y,X))
```

- True True
- True False
- False True
- False False

❓ Q40 (OOP):

Multiple inheritance resolution:

```
class M:
    def msg(self):
        return "M"
class N(M):
    def msg(self):
        return "N"
class O(M):
    pass
class P(N,O):
    pass

p=P()
print(p.msg())
```

- N (MRO chooses N first)
- M

- O
- Error

❓ Q41 (OOP):

Check bases:

```
class Base:  
    pass  
  
class Derived(Base):  
    pass  
  
print(Derived.__bases__)
```

- (<class 'main.Base'>,)
- ()
- Error
- (<class 'main.Derived'>,)

❓ Q42 (OOP):

Encapsulation with public attributes:

```
class A:  
    def __init__(self):  
        self.val=10  
  
a=A()  
print(a.val)
```

- 10
- Error
- None
- val

❓ Q43 (OOP):

Check if attribute exists using hasattr():

```
class A:  
    def __init__(self):  
        self.x=5  
  
a=A()  
print(hasattr(a, 'x'))
```

- True
- False
- Error
- None

❓ Q44 (OOP):

Polymorphism example:

```
class Animal:  
    def make_sound(self):  
        return "???"  
  
class Cat(Animal):  
    def make_sound(self):  
        return "Meow"  
  
animals=[Animal(),Cat()]  
for animal in animals:  
    print(animal.make_sound())
```

- ???\nMeow
- Meow\nMeow
- ???\n???
- Error

❓ Q45 (OOP):

Constructor arguments:

```
class A:  
    def __init__(self, x, y=10):  
        self.x=x  
        self.y=y  
  
a=A(5)  
print(a.x, a.y)
```

- 5 10

- Error
- 5 None
- None None

❓ Q46 (OOP):

Private variable name mangling:

```
class A:  
    def __init__(self):  
        self.__data=123  
  
a=A()  
print(a._A__data)
```

- 123
- __data
- Error
- None

❓ Q47 (OOP):

Class variable vs instance variable:

```
class A:  
    val=1  
  
a=A()  
a.val=2  
A.val=3  
print(a.val,A.val)
```

- 2 3
- 3 2
- 2 2
- 3 3

❓ Q48 (OOP):

Overriding **init** in a subclass:

```
class A:  
    def __init__(self,x):  
        self.x=x  
  
class B(A):
```

```
def __init__(self,x,y):
    super().__init__(x)
    self.y=y

b=B(1,2)
print(b.x,b.y)
```

- 1 2
- 2 1
- Error
- None None

❓ Q49 (OOP):

Check multiple inheritance MRO:

```
class X:
    def f(self):
        return "X"
class Y:
    def f(self):
        return "Y"
class Z(X,Y):
    pass

z=Z()
print(z.f())
```

- X (X is first in MRO)
- Y
- XY
- Error

❓ Q50 (OOP):

Using isinstance with built-in types:

```
class A:
    pass

a=A()
print(isinstance(a, object))
```

- True
- False
- Error

- None
-

Licensed for: WBS Training AG
PYDASC71e-13

✓ The Answers

Q#	Answer	Explanation
1	D	class var x=10, instance var y=20: print 10 20
2	B	A.value changed to 10 affects instance a
3	A	dict shows {'data':42} for instance a
4	A	B inherits from A, method()=A
5	A	_hidden is name-mangled, accessible as a._A_hidden=100
6	A	str returns "Value: 10"
7	A	Derived inherits from Base, greet returns "Hello from Base"
8	A	B is subclass of A => True
9	A	x1 sees class var changed to 3, x2 has instance var=2
10	A	super(). init calls A init then B init prints both
11	A	double(5)=10
12	A	empty instance has empty dict {}
13	A	Changing A.val=20 affects instance, print 20
14	A	name mangling makes _A_secret exist => True
15	A	Derived overrides greet => "Derived"
16	A	a1.val=2 (instance), others see class val=1, print 2 1 1
17	A	a.x=10 from init
18	A	str returns "A instance"
19	A	s.hello()=Hello from Super
20	A	super().greet()=A plus "B" => AB
21	A	repr returns "A(x=5)"
22	A	dict ={'data':100}
23	A	d is a Derived of Base => True
24	A	C inherits from A and B, MRO picks A.method() first => "A"
25	A	A.val=10 changed class var, a.val=20 instance var: print 10 20
26	A	super(). init sets data="A", then data+="B" => "AB"
27	A	Square(3).area=9
28	A	name mangling _A_secret_method returns 99
29	A	B inherits x=5 from A

Q#	Answer	Explanation
30	A	Calls A. init sets a="A", then b="B", print A B
31	A	Y is subclass of X => isinstance(y,X)=True
32	A	Child inherits greet from Parent => "Hi"
33	A	No x attribute => hasattr(a,'x')=False
34	A	Dog overrides sound => "Bark"
35	A	A. dict ['x']=10
36	A	A. name ="A"
37	A	A. module ="main" when run directly
38	A	repr used => "A(2)"
39	A	y is instance of Y and X => True True
40	A	MRO picks N over O => "N"
41	A	Derived. bases = (Base,)
42	A	public val=10 printed
43	A	a has x=5 => True
44	A	Animal "?":??? and Cat:"Meow"
45	A	y default=10, x=5 => print 5 10
46	A	a._A_data=123
47	A	instance val=2, class val=3
48	A	super(). init (x) sets x=1, b.y=2 => print 1 2
49	A	MRO picks X first => X
50	A	All classes inherit from object => True



3.5 - Section 5: Miscellaneous

This final section (Max. **22%** or 9x questions of the PCAP exam) covers a variety of Python features and functions that don't fit neatly into the previous categories. It includes list comprehensions, lambda functions, `map/filter/zip`, file I/O, and other utilities commonly used in everyday Python programming. Completing this quiz will round out your understanding of Python's versatile toolset, ensuring you have a broad foundation for tackling a wide range of coding tasks.

Quiz

It contains 50x questions. Please answer the questions on separate paper or in an Excel file, and then compare your answers with those in the answers section after the quiz.

Q1 (Misc):

What does the following list comprehension produce?

```
lst = [x + y for x in range(2) for y in range(3)]  
print(lst)
```

- [0, 1, 2]
- [0, 1, 2, 1, 2, 3]
- [0, 1, 2, 1, 2, 3]
- [2, 3, 4, 3, 4, 5]

Q2 (Misc):

What is the output of the following code?

```
def func():  
    return lambda x: x * x  
  
square = func()  
print(square(5))
```

- `lambda x: x * x`
- [25]
- 25
- `x * x`

Q3 (Misc):

What does the `map` function do in Python?

```
numbers = [1, 2, 3, 4]
result = list(map(lambda x: x * 2, numbers))
print(result)
```

- Filters the list based on a condition.
- Applies a function to each item of the list.
- Sorts the list.
- Reverses the list.

❓ Q4 (Misc):

What is the output of the following code?

```
nums = [1, 2, 3, 4]
filtered = list(filter(lambda x: x % 2 == 0, nums))
print(filtered)
```

- [1, 3]
- [2, 4]
- [1, 2, 3, 4]
- []

❓ Q5 (Misc):

What does the `zip` function do?

```
a = [1, 2, 3]
b = ['a', 'b']
z = list(zip(a, b))
print(z)
```

- Combines lists into a single list.
- Pairs elements from multiple lists into tuples.
- Merges dictionaries.
- Sorts the lists.

❓ Q6 (Misc):

What is the result of the following code?

```
s = "Hello, World!"
print(s[::-1])
```

- "olleH ,dlroW!"
- "Hello, World!"
- "Hello, Wrd!"
- Error

❓ Q7 (Misc):

What does the `any` function do?

```
lst = [0, False, 5]
print(any(lst))
```

- Returns `True` only if all elements are `True`.
- Returns `True` if at least one element is `True`.
- Returns `False` always.
- Raises an exception.

❓ Q8 (Misc):

What is the output of the following code?

```
lst = [1, 2, 3]
print(*lst)
```

- 1 2 3
- [1, 2, 3]
- "1 2 3"
- *1 *2 *3

❓ Q9 (Misc):

What does the `enumerate` function do?

```
for index, value in enumerate(['a', 'b', 'c']):
    print(index, value)
```

- Iterates only over the values.
- Provides both index and value for each iteration.
- Sorts the list before iterating.
- Creates a dictionary from the list.

❓ Q10 (Misc):

What is the output of the following code?

```
def outer():
    x = "outer"
    def inner():
        nonlocal x
        x = "inner"
    inner()
print(x)

outer()
```

- "outer"
- "inner"
- Error due to `nonlocal`
- Nothing

❓ Q11 (Misc):

What does the `sorted` function return?

```
lst = [3, 1, 4, 2]
sorted_lst = sorted(lst)
print(sorted_lst)
```

- [1, 2, 3, 4]
- [4, 3, 2, 1]
- [3, 1, 4, 2]
- None

❓ Q12 (Misc):

What is the output of the following code?

```
s = "Python"
print(s.startswith("Py"))
```

- True
- False
- "Py"
- Error

❓ Q13 (Misc):

What does the `reversed` function do?

```
lst = [1, 2, 3]
rev = list(reversed(lst))
print(rev)
```

- Sorts the list.
- Returns a reversed iterator.
- Removes duplicates.
- Raises an exception.

❓ Q14 (Misc):

What is the output of the following code?

```
lst = [1, 2, 3]
print(lst.pop())
print(lst)
```

- 3 followed by [1, 2]
- 1 followed by [2, 3]
- 3 followed by [1, 2, 3]
- Raises an IndexError

❓ Q15 (Misc):

What does the `any` function return for an empty iterable?

```
lst = []
print(any(lst))
```

- False
- True
- Raises an exception
- None

❓ Q16 (Misc):

What is the result of the following code?

```
s = "Data Science"
print(s.replace(" ", ""))
```

- "DataScience"
- "Data Science"

- "Data_Science"
- Raises an exception

❓ Q17 (Misc):

What does the `max` function do?

```
lst = [1, 5, 3]
print(max(lst))
```

- Returns the largest item.
- Returns the smallest item.
- Returns the sum.
- Returns the average.

❓ Q18 (Misc):

What is the output of the following code?

```
s = "Mississippi"
print(s.count("iss"))
```

- 2
- 1
- 3
- 4

❓ Q19 (Misc):

What does the `join` method do?

```
lst = ['a', 'b', 'c']
print('-'.join(lst))
```

- Concatenates the list elements separated by '-'.
- Splits the string by '-'.
- Converts list to a tuple.
- Raises an exception.

❓ Q20 (Misc):

What does the `strip` method do?

```
s = "Hello World "
```

```
print(s.strip())
```

- Removes leading and trailing whitespace.
- Removes all whitespace.
- Removes only leading whitespace.
- Removes only trailing whitespace.

❓ Q21 (Misc):

What is the output of the following code?

```
s = "abc123"  
print(s.isdigit())
```

- False
- True
- Raises an exception
- None

❓ Q22 (Misc):

What does the `format` method do?

```
name = "Alice"  
age = 30  
print("Name: {}, Age: {}".format(name, age))
```

- Inserts variables into the string placeholders.
- Concatenates strings.
- Reverses the string.
- Raises an exception.

❓ Q23 (Misc):

What does "`PYTHON`".islower() return?

- True
- False
- None
- Raises an exception

❓ Q24 (Misc):

What does the following code print?

```
s = "banana"  
print(s[::-2])
```

- "bnn"
- "anana"
- "banana"
- "baaa"

❓ Q25 (Misc):

What is the result of the following code?

```
s = "Hello World"  
print(s.partition(" "))
```

- ('Hello', ' ', 'World')
- ['Hello', 'World']
- ('HelloWorld',)
- Raises an exception

❓ Q26 (Misc):

What does the `lstrip` method do?

```
s = "Hello"  
print(s.lstrip())
```

- Removes leading whitespace.
- Removes trailing whitespace.
- Removes all whitespace.
- Does nothing.

❓ Q27 (Misc):

What does `"123".isnumeric()` return?

- True
- False
- None
- Raises an exception

❓ Q28 (Misc):

What is the output of the following code?

```
s = "hello"  
print(s.capitalize())
```

- "Hello"
- "HELLO"
- "hello"
- "hELLO"

❓ Q29 (Misc):

What does the `index` method do?

```
s = "hello"  
print(s.index("e"))
```

- Returns the index of the first occurrence of "e".
- Returns a list of all indices of "e".
- Replaces "e" with another character.
- Raises an exception if "e" is not found.

❓ Q30 (Misc):

What does the `startswith` method check?

```
s = "Python"  
print(s.startswith("Py"))
```

- Whether the string starts with the specified substring.
- Whether the string contains the substring.
- Whether the string ends with the substring.
- Reverses the string.

❓ Q31 (Misc):

What is the output of the following code?

```
s = "Data"  
print(s.isupper())
```

- **False**
- **True**
- **None**

- Raises an exception

❓ Q32 (Misc):

What does the `replace` method do?

```
s = "foo bar foo"
print(s.replace("foo", "baz"))
```

- Replaces all occurrences of "foo" with "baz".
- Reverses the string.
- Removes "foo" from the string.
- Raises an exception if "foo" is not found.

❓ Q33 (Misc):

What is the output of the following code?

```
s = "Hello World"
print(s.find("World"))
```

- 6
- -1
- 11
- Raises an exception

❓ Q34 (Misc):

What does the `rjust` method do?

```
s = "hi"
print(s.rjust(5, '*'))
```

- Pads the string on the left with '*' to make it length 5.
- Pads the string on the right with '*' to make it length 5.
- Centers the string with '*' padding.
- Removes '*' from the string.

❓ Q35 (Misc):

What is the output of the following code?

```
s = "Hello"  
print(s[1:4])
```

- "ell"
- "Hel"
- "lo"
- "ello"

❓ Q36 (Misc):

What does the `join` method require as its argument?

```
",".join(['a', 'b', 'c'])
```

- An iterable (like a list or tuple) of strings.
- Only a list of integers.
- A single string.
- Any object.

❓ Q37 (Misc):

What is the output of the following code?

```
s = "Data Science"  
print(s.count("a"))
```

- 1
- 2
- 0
- 3

❓ Q38 (Misc):

What does the `split` method without arguments do?

```
s = "Hello World"  
print(s.split())
```

- Splits the string on whitespace.
- Splits the string on every character.
- Does nothing.
- Raises an exception.

?

Q39 (Misc):

What is the output of the following code?

```
s = "Hello"  
print(s.endswith("lo"))
```

- True
- False
- None
- Raises an exception

?

Q40 (Misc):

What does the `isalpha` method check?

```
s = "Hello"  
print(s.isalpha())
```

- Whether all characters in the string are alphabetic.
- Whether the string contains any digits.
- Whether the string is empty.
- Whether the string starts with a capital letter.

?

Q41 (Misc):

What is the output of the following code?

```
s = "Hello"  
print(s.index("l"))
```

- 2 (first occurrence of "l")
- 3
- -1
- Raises an exception

?

Q42 (Misc):

What does the `format` method allow you to do?

```
name = "Alice"  
age = 30  
print("Name: {}, Age: {}".format(name, age))
```

- Insert variables into a string at specified placeholders.
- Reverse the string.
- Concatenate without spaces.
- Remove variables from the string.

❓ Q43 (Misc):

What is the output of the following code?

```
s = "Hello"  
print(s.islower())
```

- False
- True
- None
- Raises an exception

❓ Q44 (Misc):

What does the `capitalize` method do?

```
s = "hello world"  
print(s.capitalize())
```

- Capitalizes the first character and lowers the rest.
- Capitalizes all characters.
- Capitalizes the first character of each word.
- Does nothing.

❓ Q45 (Misc):

What does the `join` method do when used with an empty string?

```
s = ""  
lst = ['a', 'b', 'c']  
print(s.join(lst))
```

- "abc"
- "a b c"
- "a,b,c"
- Raises an exception

❓ Q46 (Misc):

What is the output of the following code?

```
s = "Hello World"  
print(s.replace("World", "Python"))
```

- "Hello Python"
- "HelloWorld"
- "Hello WorldPython"
- Raises an exception

❓ Q47 (Misc):

What does the `startswith` method return if the string starts with the given substring?

```
s = "Data Science"  
print(s.startswith("Data"))
```

- True
- False
- None
- Raises an exception

❓ Q48 (Misc):

What does `" ".isspace()` return?

- True
- False
- None
- Raises an exception

❓ Q49 (Misc):

What is the output of the following code?

```
s = "Python"  
print(s[1:100])
```

- "ython"
- Raises an IndexError
- "ython" followed by empty strings
- "Python"

❓ Q50 (Misc):

What does the `count` method do?

```
s = "banana"  
print(s.count("ana"))
```

- Counts the number of non-overlapping occurrences of "ana".
 - Counts all overlapping occurrences of "ana".
 - Replaces "ana" with another substring.
 - Raises an exception if "ana" is not found.
-

Licensed for: WBS Training AG
PYDASC71e-13

✓ The Answers

Q#	Answer	Explanation
1	C	The list comprehension iterates over <code>x</code> in range(2) and <code>y</code> in range(3), producing [0+0, 0+1, 0+2, 1+0, 1+1, 1+2] which is [0,1,2,1,2,3].
2	C	The lambda function returns the square of 5, which is 25.
3	B	<code>map</code> applies the lambda function to each element, resulting in [2,4,6,8].
4	B	<code>filter</code> selects even numbers, resulting in [2,4].
5	B	<code>zip</code> pairs elements from <code>a</code> and <code>b</code> into tuples, stopping at the shortest list, resulting in [(1,1), (2,2)].
6	A	Slicing with <code>[::-1]</code> reverses the string, resulting in "!dlroW ,olleH".
7	B	<code>any</code> returns <code>True</code> if at least one element is <code>True</code> ; here, 5 is truthy.
8	C	The list <code>[0,1,2]</code> is printed by unpacking with <code>*</code> .
9	A	<code>enumerate</code> provides both index and value, printing "0 a", "1 b", "2 c".
10	B	<code>outer</code> defines <code>x = "outer"</code> , <code>inner</code> changes <code>x</code> to "inner" using <code>nonlocal</code> , so <code>x</code> becomes "inner".
11	A	<code>dir()</code> lists all attributes and methods of the module.
12	B	<code>sys.path</code> is a list of directories the interpreter searches for modules.
13	A	<code>math.ceil(3.7)</code> returns 4.
14	A	The list comprehension multiplies "abc" by 3, resulting in "abcabcabc".
15	A	<code>reversed</code> returns a reversed iterator; converting to list gives [3,2,1].
16	A	<code>s[::-1]</code> reverses the string, resulting in "!dlroW ,olleH".
17	A	<code>any([])</code> returns <code>False</code> because the iterable is empty.
18	A	The <code>replace</code> method removes spaces, resulting in "DataScience".
19	A	<code>max(lst)</code> returns the largest item, which is 4.
20	A	"Mississippi" contains "iss" twice.
21	A	<code>join</code> concatenates the list with '-', resulting in "a-b-c".
22	A	<code>strip</code> removes leading and trailing whitespace, resulting in "spaced".
23	A	<code>s.isdigit()</code> returns <code>False</code> because the string contains letters.
24	A	Multiplying "abc" by 3 results in "abcabcabc".
25	A	<code>"Hello" * 0</code> returns an empty string "".
26	A	<code>map</code> applies a function to each item, resulting in [2,4,6,8].
27	A	Reversed iterator converted to list gives [3,2,1].

Q#	Answer	Explanation
28	A	<code>any(1st)</code> returns <code>True</code> because 5 is truthy.
29	A	Slicing beyond the string length stops at the end without error.
30	A	<code>count("ana")</code> counts non-overlapping "ana", which appears once in "banana".
31	A	<code>enumerate</code> provides both index and value, resulting in [0,1,2].
32	A	The lambda function adds 2 to each even number, resulting in [4,16].
33	A	<code>reversed</code> returns a reversed iterator, resulting in [3,2,1].
34	A	<code>any(1st)</code> returns <code>True</code> because 5 is truthy.
35	A	The list comprehension multiplies "abc" by 3, resulting in "abcabcabc".
36	A	Slicing with <code>[::-1]</code> reverses the string.
37	A	The lambda function squares 5, resulting in 25.
38	A	<code>map</code> applies a function to each item, resulting in [2,4,6,8].
39	A	<code>filter</code> selects even numbers, resulting in [2,4].
40	A	<code>zip</code> pairs elements from <code>a</code> and <code>b</code> into tuples, stopping at the shortest list, resulting in <code>[(1,'a'), (2,'b')]</code> .
41	A	Slicing with <code>[::-1]</code> reverses the string, resulting in "!dlroW ,olleH".
42	A	<code>any(1st)</code> returns <code>True</code> because 5 is truthy.
43	A	Unpacking with <code>*</code> prints elements separated by space: "1 2 3".
44	A	<code>enumerate</code> provides both index and value, resulting in [0,1,2].
45	A	The lambda function adds 2 to each even number, resulting in [4,16].
46	A	<code>join</code> concatenates the list with '-', resulting in "a-b-c".
47	A	Slicing with <code>[::-1]</code> reverses the string.
48	A	<code>any(1st)</code> returns <code>True</code> because 5 is truthy.
49	A	<code>map</code> applies a function to each item, resulting in [2,4,6,8].
50	A	<code>filter</code> selects even numbers, resulting in [2,4].

Licensed for: WBS Training AG
PYDASC71e-13



4.1 Exam 1

- **Exam Name:** PCAP™ – Certified Associate Python Programmer - **PCAP-31-03**
- **Duration:** 65 minutes
- **Passing Score:** 70%

Exam

It contains 40x questions. Please answer the questions on separate paper or in an Excel file, and then compare your answers with those in the answers section after the quiz.

Q1:

What is the output of the following code?

```
import math

x = 4.7
y = math.floor(x) + math.ceil(x)
print(y)
```

- 8
- 10
- 9
- 5

Q2:

What does the following code print?

```
import sys
print(sys.path)
```

- The current Python search path for modules as a list of strings
- The current working directory only
- The list of all built-in modules
- The directory of the Python interpreter executable only

Q3:

What is the behavior of the following code?

```
import random
```

```
lst = [10, 20, 30, 40]
val = random.choice(lst)
print(val)
```

- It will always print 10
- It will print a random element from the list [10,20,30,40]
- It will raise a TypeError
- It will print a list of all elements

❓ Q4:

What is printed?

```
from math import ceil, floor

x = 3.5
print(ceil(x), floor(x))
```

- 3 3
- 4 3
- 3.5 3.5
- 3 4

❓ Q5:

Assume a package structure as shown and the following code:

```
import mypkg
print(mypkg.__name__)
```

What is the likely output?

- mypkg
- main
- module_a
- It raises ImportError

❓ Q6:

In Python modules, there is no enforced private variable system, but what naming convention is commonly used to indicate a "private" variable?

- Prepending a variable name with a single underscore (e.g. `_secret_value`)
- Using all uppercase letters
- Python enforces private variables by syntax

- Using a trailing underscore

❓ Q7:

What happens when you run this code?

```
try:  
    x = 10 / 0  
except ZeroDivisionError:  
    print("Cannot divide by zero!")
```

- It prints "Cannot divide by zero!"
- It raises a ZeroDivisionError without handling
- It prints nothing
- It prints "Cannot divide by zero!" and then raises another error

❓ Q8:

What will happen when this code runs?

```
def process_value(val):  
    assert val >= 0, "Value must be non-negative"  
    return val * 2  
  
print(process_value(-1))
```

- It prints -2
- It prints 2
- It raises an AssertionError with the message "Value must be non-negative"
- It raises a ValueError

❓ Q9:

What is printed?

```
class MyError(Exception):  
    pass  
  
try:  
    raise MyError("Custom error occurred")  
except MyError as e:  
    print(e)
```

- MyError
- Custom error occurred

- Nothing is printed
- e

❓ Q10:

Assume 'nonexistent_file.txt' does not exist. What will be the output?

```
try:  
    f = open('nonexistent_file.txt', 'r')  
except FileNotFoundError as e:  
    print("File not found:", e.args)
```

- It prints "File not found:" followed by a tuple containing the error message
- It prints "File not found:" only
- It prints nothing and no exception is raised
- It raises FileNotFoundError without printing anything

❓ Q11:

What is the output?

```
try:  
    x = int("abc")  
except ValueError:  
    print("Value error encountered!")  
else:  
    print("No error found!")  
finally:  
    print("Cleanup complete!")
```

- "Value error encountered!" followed by "Cleanup complete!"
- "No error found!" followed by "Cleanup complete!"
- "Value error encountered!" only
- "Cleanup complete!" only

❓ Q12:

What is printed?

```
s = "Hello, World!"  
print(s[7:12])
```

- World
- Worl
- orld

- Hello

❓ Q13:

What does the code print?

```
s = "Python3.12"  
print(s.isalnum())
```

- True
- False
- Raises AttributeError
- It depends on the system

❓ Q14:

If `s = "Hello"`, what does the following print?

```
s = "Hello"  
x = ord(s[1])  
print(x)
```

- 101
- 69
- 104
- 111

❓ Q15:

What is printed?

```
s = "abcdef"  
print("z" in s)
```

- True
- False
- 'z'
- It raises a NameError

❓ Q16:

What is printed?

```
s = " Trim me "
```

```
print(s.strip())
```

- " Trim me "
- "Trim me"
- "Trim me "
- " Trim me"

❓ Q17:

What does this print?

```
s = "banana"  
print(s.find("na"))
```

- 2
- 3
- 1
- -1

❓ Q18:

What does this print?

```
s = "hello"  
print(s.upper().startswith("H"))
```

- True
- False
- "H"
- Raises AttributeError

❓ Q19:

What does `s.split()` return?

```
s = "Hello World"  
print(s.split())
```

- ["Hello", "World"]
- "Hello World"
- ["Hello World"]
- ("Hello", "World")

?

Q20:

What is printed?

```
class MyClass:  
    pass  
  
obj = MyClass()  
print(type(obj))
```

- <class 'main.MyClass'>
- MyClass
- type
- <class 'MyClass'>

?

Q21:

What is printed?

```
class MyClass:  
    def __init__(self, x):  
        self.x = x  
  
obj = MyClass(10)  
print(obj.x)
```

- 10
- None
- MyClass
- Error at runtime

?

Q22:

What is printed?

```
class A:  
    def __init__(self):  
        self.value = 42  
  
a = A()  
print(a.__dict__)
```

- {'value': 42}
- {}
- {'dict': {}}

- An AttributeError

?

Q23:

What is printed?

```
class A:  
    def method(self):  
        return "A"  
  
class B(A):  
    def method(self):  
        return "B"  
  
obj = B()  
print(obj.method())
```

- A
- B
- Error due to no inheritance
- None

?

Q24:

What is printed?

```
class A:  
    def __init__(self, val):  
        self.__val = val  
  
    def get_val(self):  
        return self.__val  
  
a = A(10)  
print(a.get_val())
```

- 10
- Error due to private variable
- None
- __val

?

Q25:

What does this print?

```
class A:  
    pass
```

```
a = A()  
print(hasattr(a, 'x'))
```

- True
- False
- Error
- None

❓ Q26:

What is printed?

```
class A:  
    def __str__(self):  
        return "Instance of A"  
  
a = A()  
print(a)
```

- Instance of A
- <main.A object at 0x...>
- A
- Raises TypeError

❓ Q27:

What is printed?

```
class Base:  
    pass  
  
class Derived(Base):  
    pass  
  
d = Derived()  
print(isinstance(d, Base))
```

- True
- False
- Error
- None

❓ Q28:

What is printed?

```
class X:  
    val = 5  
  
x1 = X()  
x2 = X()  
x1.val = 10  
print(X.val, x2.val)
```

- 5 5
- 10 5
- 10 10
- 5 10

❓ Q29:

What is printed?

```
class MyClass:  
    def __init__(self):  
        print("Constructor called")  
  
obj = MyClass()
```

- Constructor called
- Nothing
- Error
- None

❓ Q30:

What is printed?

```
class A:  
    def __init__(self, x):  
        self.x = x  
  
    def display(self):  
        print(self.x)  
  
a = A(5)  
a.display()
```

- 5
- self.x
- x

- None

❓ Q31:

What is printed?

```
numbers = [1, 2, 3, 4]
squares = [n*n for n in numbers]
print(squares)
```

- [1, 4, 9, 16]
- (1, 4, 9, 16)
- {1, 4, 9, 16}
- 1 4 9 16

❓ Q32:

What is printed?

```
func = lambda x: x + 1
print(func(5))
```

- 6
- x + 1
- func
- None

❓ Q33:

What is printed?

```
numbers = [1, 2, 3]
result = list(map(lambda x: x*x, numbers))
print(result)
```

- [1, 4, 9]
- (1, 4, 9)
- {1, 4, 9}
- map object

❓ Q34:

What is printed?

```
numbers = [1, 2, 3, 4]
filtered = list(filter(lambda x: x%2 == 0, numbers))
print(filtered)
```

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

❓ Q35:

What is printed?

```
letters = ['a', 'b', 'c']
numbers = [1, 2, 3]
zipped = list(zip(letters, numbers))
print(zipped)
```

- [('a', 1), ('b', 2), ('c', 3)]
- [('a', 'b', 'c'), (1, 2, 3)]
- [['a', 1], ['b', 2], ['c', 3]]
- ['a1', 'b2', 'c3']

❓ Q36:

What does this code do?

```
with open('data.txt', 'w') as f:
    f.write("Hello")
```

- Creates or overwrites data.txt with the content "Hello"
- Appends "Hello" to data.txt
- Opens data.txt in read mode and prints "Hello"
- Raises an exception

❓ Q37:

What is the type of **content**?

```
with open('data.txt', 'rb') as f:
    content = f.read()
print(type(content))
```

- str
- bytes
- list
- bytearray

❓ Q38:

What is printed?

```
data = [1,2,3,4]
res = [x for x in data if x > 2]
print(res)
```

- [3,4]
- [1,2,3,4]
- [2,3,4]
- [1,2]

❓ Q39:

What is printed?

```
def make_multiplier(n):
    def multiplier(x):
        return x * n
    return multiplier

double = make_multiplier(2)
print(double(5))
```

- 10
- 7
- Error
- 2

❓ Q40:

What is printed?

```
data = bytearray(b'\x00\x01\x02')
data[1] = 255
print(data)
```

- bytearray(b'\x00\xff\x02')
- bytearray(b'\x00\x01\x02')

- b"\x00\xff\x02"
 - [0, 255, 2]
-

Licensed for: WBS Training AG
PYDASC71e-13

✓ The Answers

Q#	Answer	Explanation
1	C	<code>floor(4.7)=4, ceil(4.7)=5, sum=9</code>
2	A	<code>sys.path</code> is a list of module search paths
3	B	<code>random.choice</code> returns a random element
4	B	<code>ceil(3.5)=4, floor(3.5)=3, so "4 3"</code>
5	A	<code>name</code> of imported package is its own name, here "mypkg"
6	A	Convention: a leading underscore indicates "private" in Python
7	A	<code>ZeroDivisionError</code> caught, prints message
8	C	<code>val < 0</code> triggers <code>AssertionError</code> with given message
9	B	Raises <code>MyError</code> , prints "Custom error occurred"
10	A	Caught <code>FileNotFoundException</code> prints "File not found:" and args
11	A	<code>ValueError</code> in try -> except prints message, finally prints cleanup
12	A	<code>s[7:12]</code> in "Hello, World!" = "World"
13	B	"Python3.12" not alnum due to '.'
14	A	<code>s[1]='e', ord('e')=101</code>
15	B	'z' not in "abcdef" => False
16	B	<code>strip</code> removes whitespace, leaving "Trim me"
17	A	"na" first in "banana" at index 2
18	A	"HELLO".startswith("H") = True
19	A	"Hello World".split() -> ["Hello", "World"]
20	A	<code>type(obj)=<class 'main.MyClass'></code>
21	A	<code>obj.x = 10</code> printed
22	A	<code>dict</code> shows {'value':42}
23	B	Overridden method returns "B"
24	A	<code>get_val</code> returns 10
25	B	<code>hasattr(a,'x')=False</code> as 'x' not defined
26	A	<code>str</code> returns "Instance of A"
27	A	Derived inherits Base, <code>isinstance(d,Base)=True</code>
28	A	<code>X.val=5</code> remains, <code>x2.val =5</code> from class var
29	A	Constructor prints "Constructor called"

Q#	Answer	Explanation
30	A	a.display() prints self.x=5
31	A	[n*n for n in [1,2,3,4]] = [1,4,9,16]
32	A	lambda x:x+1 for x=5 -> 6
33	A	map with x*x -> [1,4,9]
34	A	filter even from [1,2,3,4] -> [2,4]
35	A	zip(['a','b','c'],[1,2,3]) -> [('a',1),('b',2),('c',3)]
36	A	'w' mode creates/overwrites file with "Hello"
37	B	'rb' mode returns bytes
38	A	[x for x in [1,2,3,4] if x>2] = [3,4]
39	A	closure returns multiplier: double(5)=10
40	A	bytearray modified to b'\x00\xff\x02'

Licensed for: WBS Training AG
PYDASC71e-13



4.2 Exam 2

- **Exam Name:** PCAP™ – Certified Associate Python Programmer - **PCAP-31-03**
- **Duration:** 65 minutes
- **Passing Score:** 70%

Exam

It contains 40x questions. Please answer the questions on separate paper or in an Excel file, and then compare your answers with those in the answers section after the quiz.

Q1:

Consider the following code snippet:

```
import math

val = -3.7
res = math.floor(val) + math.ceil(val)
print(res)
```

What is the output?

- -7
- -3
- -7
- 0
- -4

Q2:

Given this code:

```
import random

random.seed(10)
values = [1, 2, 3, 4]
print(random.choice(values))
```

What will it print?

- 2
- 1
- 3

- The output is unpredictable

?

Q3:

What does this code print?

```
import sys

sys.path.append("/custom/path")
print("/custom/path" in sys.path)
```

- True
- False
- It raises an ImportError
- It prints the entire sys.path

?

Q4:

Consider the structure of a package `mypkg` containing an `__init__.py` with a variable defined:

`__init__.py`:

```
myvar = 42
```

Code:

```
import mypkg
print(mypkg.myvar)
```

What will this print?

- 42
- It will raise AttributeError
- It will print `mypkg`
- It will print `__main__`

?

Q5:

Analyze the following code:

```
from math import ceil as c, floor as f

x = 2.9
print(c(x) - f(x))
```

What will it print?

- 2
- 1.9
- 1
- 3

❓ Q6:

What does the following code print?

```
import random

lst = [10,20,30]
print(random.sample(lst, 2))
```

- [10,20,30]
- 10,20
- A random 2-element list chosen from [10,20,30], e.g. [20,30]
- An empty list

❓ Q7:

What does the following code do?

```
try:
    x = int("3.14")
except ValueError as e:
    print("ValueError:", e.args)
else:
    print("No error")
finally:
    print("Done")
```

- It prints "ValueError:" and the args tuple, then "Done"
- It prints "No error" then "Done"
- It raises a ValueError without handling
- It prints "Done" only

❓ Q8:

What will be printed?

```
try:
    raise ZeroDivisionError("Division by zero!")
except (ValueError, TypeError) as e:
```

```
    print("Handled Value or Type error")
except ZeroDivisionError as zde:
    print("ZeroDivision handled:", zde)
```

- Handled Value or Type error
- ZeroDivision handled: Division by zero!
- It raises ZeroDivisionError
- Nothing is printed

❓ Q9:

Given:

```
class MyCustomError(Exception):
    pass

try:
    raise MyCustomError("Something went wrong")
except MyCustomError as err:
    print("Caught:", err)
```

What is printed?

- Caught: MyCustomError
- Caught: Something went wrong
- Caught: err
- Nothing is printed

❓ Q10:

Consider the code:

```
def process(val):
    assert isinstance(val, int), "val must be an integer"
    return val * 2

print(process("hello"))
```

What happens?

- Prints "hellohello"
- Prints 0
- Raises AssertionError with message "val must be an integer"
- Raises TypeError

❓ Q11:

What does the following print?

```
s = "Hello\nWorld"  
print(s.count('l'))
```

- 2
- 3
- 3
- 4

❓ Q12:

What is printed?

```
s = "Python3.12"  
print(s.isalpha())
```

- True
- False (because of digits and '.')
- Raises AttributeError
- Depends on the environment

❓ Q13:

What does this print?

```
s = "Hello"  
print(s[1:4])
```

- ell
- Hel
- llo
- ello

❓ Q14:

Given the code:

```
s = "ABCDE"  
print(s[::2])
```

What does it print?

- ACE
- ABC
- ADE
- BCD

❓ Q15:

What will this print?

```
s = "banana"  
print(s.rfind("na"))
```

- 1
- 4
- 2
- -1

❓ Q16:

What does this print?

```
s = " test "  
print(s.lstrip())
```

- "test "
- " test"
- "test"
- " test "

❓ Q17:

Consider:

```
s = "abc123"  
print(s.isalnum())
```

- True
- False
- Raises TypeError
- Depends on locale

❓ Q18:

What does this print?

```
s = "HelloWorld"  
print("low" in s)
```

- True
- False
- low
- Raises ValueError

❓ Q19:

What does this print?

```
s = "Hello, World!"  
print(s.replace("World", "Universe"))
```

- Hello, Universe!
- Hello, World!
- Hello,
- Universe

❓ Q20:

What does this code output?

```
class MyClass:  
    x = 10  
    def __init__(self, y):  
        self.y = y  
  
obj = MyClass(20)  
print(obj.x, obj.y)
```

- 10 20
- 20 10
- Error
- None None

❓ Q21:

What is printed?

```
class A:  
    val = 5
```

```
a = A()  
A.val = 10  
print(a.val)
```

- 5
- 10
- Error
- None

❓ Q22:

Analyze the code:

```
class A:  
    def __init__(self):  
        self._hidden = 42  
  
a = A()  
print(a._A_hidden)
```

- 42 (name mangling: _A_hidden)
- Error (private variable)
- None
- _A_hidden

❓ Q23:

What is printed?

```
class A:  
    def method(self):  
        return "A"  
  
class B(A):  
    pass  
  
b = B()  
print(b.method())
```

- A
- B
- Error
- None

❓ Q24:

What is printed?

```
class A:  
    def __init__(self, x):  
        self.x = x  
  
    def __str__(self):  
        return f"Value: {self.x}"  
  
obj = A(5)  
print(obj)
```

- Value: 5
- 5
- A object
- Raises TypeError

❓ Q25:

Consider the code:

```
class A:  
    def __init__(self):  
        self.data = 100  
  
a = A()  
print(a.__dict__)
```

- {'data': 100}
- {}
- Raises AttributeError
- **dict** is empty

❓ Q26:

What does the code print?

```
class Base:  
    def greet(self):  
        return "Hello from Base"  
  
class Derived(Base):  
    def greet(self):  
        return "Hello from Derived"  
  
d = Derived()  
print(isinstance(d, Base), d.greet())
```

- True Hello from Derived
- True Hello from Base
- False Hello from Derived
- False Hello from Base

❓ Q27:

Given:

```
class A:  
    pass  
  
class B(A):  
    pass  
  
b = B()  
print(issubclass(B, A))
```

- True
- False
- Raises TypeError
- None

❓ Q28:

Consider:

```
class A:  
    val = 1  
  
a1 = A()  
a2 = A()  
a1.val = 2  
A.val = 3  
print(a1.val, a2.val)
```

- 2 3
- 2 2
- 3 3
- 1 1

❓ Q29:

What is printed?

```
class A:  
    def __init__(self):  
        print("A init")  
  
class B(A):  
    def __init__(self):  
        super().__init__()  
        print("B init")  
  
b = B()
```

- A init \n B init
- B init \n A init
- A init only
- B init only

❓ Q30:

Given:

```
class A:  
    def __init__(self, x):  
        self.x = x  
    def show(self):  
        return self.x * 2  
  
a = A(4)  
print(a.show())
```

- 8
- 4
- Error
- None

❓ Q31:

What does this print?

```
lst = [x**2 for x in range(5) if x%2 == 1]  
print(lst)
```

- [1, 9]
- [0,1,4,9,16]
- [1,4]
- []

❓ Q32:

Analyze:

```
func = lambda x, y: x*y  
print(func(3,4))
```

- 12
- 7
- x*y
- Error

❓ Q33:

What is printed?

```
nums = [1,2,3,4]  
result = list(map(lambda x: x+10, filter(lambda x: x%2==0, nums)))
```

```
print(result)
```

- [12,14]
- [11,12,13,14]
- [2,4]
- [10,10]

❓ Q34:

What is printed?

```
nums = [1,2,3,4,5]
res = [x*x for x in nums if x>2 and x<5]
print(res)
```

- [9,16] (for x=3,4)
- [1,4,9,16,25]
- [9,16,25]
- []

❓ Q35:

What happens here?

```
letters = ['a','b']
numbers = [1,2,3]
z = list(zip(letters, numbers))
print(z)
```

- [('a',1),('b',2)]
- [('a',1),('b',2), (None,3)]
- [('a','b'),(1,2,3)]
- Error

❓ Q36:

What does this code do?

```
with open('output.txt', 'a') as f:
    f.write("Data\n")
```

- Appends "Data\n" to output.txt
- Writes "Data\n" overwriting file
- Raises FileNotFoundError

- Opens in read mode

?

Q37:

What will be the type of **data**?

```
with open('input.bin','rb') as f:  
    data = f.read(5)  
print(type(data))
```

- <class 'bytes'>
- <class 'str'>
- <class 'bytearray'>
- <class 'list'>

?

Q38:

What does this print?

```
data = [x for x in range(10) if x%3==0]  
print(data)
```

- [0,3,6,9]
- [3,6,9]
- [0,3,6]
- []

?

Q39:

What does this code print?

```
def make_power(n):  
    def power(x):  
        return x**n  
    return power  
  
cube = make_power(3)  
print(cube(2))
```

- 8
- 6
- 9
- Error

?

Q40:

What is printed?

```
arr = bytearray(b'\x01\x02\x03')
arr[2] = 0xFF
print(arr)
```

- bytearray(b'\x01\x02\xff')
 - bytearray(b'\x01\x02\x03')
 - b'\x01\x02\xff'
 - [1,2,255]
-
-

Licensed for: WBS Training AG
PYDASC71e-13

✓ The Answers

Q#	Answer	Explanation
1	C	<code>floor(-3.7)=-4, ceil(-3.7)=-3 sum=-7</code>
2	A	<code>random.seed(10)</code> makes choice from [1,2,3,4] = 2
3	A	/custom/path added to <code>sys.path</code> , print True
4	A	<code>init.py</code> sets <code>myvar=42</code> accessible after import
5	C	<code>ceil(2.9)=3, floor(2.9)=2, 3-2=1</code>
6	C	<code>random.sample(lst,2)</code> returns a 2-element random list
7	A	<code>int("3.14") => ValueError</code> , prints error args, then Done
8	B	matches <code>ZeroDivisionError</code> except, prints message
9	B	Caught: Something went wrong (exception message)
10	C	<code>assert</code> fails, raises <code>AssertionError</code>
11	C	"Hello\nWorld" has 3 'l's total
12	B	s="Python3.12" has digits and '.', not <code>isalpha</code>
13	A	s[1:4]="ell" from "Hello"
14	A	s="ABCDE"[::2]="ACE"
15	B	"banana". <code>rfind("na")=4</code>
16	A	<code>lstrip</code> removes left spaces => "test "
17	A	"abc123". <code>isalnum()</code> =True
18	A	"loW" in "HelloWorld"? substring "loW" at s[3:6]
19	A	replace World with Universe => "Hello, Universe!"
20	A	obj.x=10(class), obj.y=20(instance), print "10 20"
21	B	Changing A.val=10 affects a, so prints 10
22	A	<code>_hidden</code> accessible via <code>_A_hidden=42</code>
23	A	B inherits from A, method returns "A"
24	A	<code>str</code> returns "Value: 5"
25	A	<code>dict={'data':100}</code>
26	A	<code>isinstance(d,Base)=True</code> , greet=overridden in Derived
27	A	B is subclass of A => True
28	A	a1.val=2 (instance), A.val=3 changes class val => a2 sees 3
29	A	<code>super().init</code> calls A init first, then prints B init

Q#	Answer	Explanation
30	A	show() returns 8
31	A	Odd numbers:1,3 their squares:1,9
32	A	$3*4=12$
33	A	even:2,4 => +10 => [12,14]
34	A	$x=3,4 \Rightarrow 9,16$
35	A	zip stops at shortest => [('a',1),('b',2)]
36	A	'a' mode appends "Data\n"
37	A	read in 'rb' mode returns bytes
38	A	multiples of 3 in range(10):0,3,6,9
39	A	closure cube(2)=8
40	A	bytearray changed third element to 0xFF

Licensed for: WBS Training AG
PYDASC71e-13



4.3 Exam 3

- **Exam Name:** PCAP™ – Certified Associate Python Programmer - **PCAP-31-03**
- **Duration:** 65 minutes
- **Passing Score:** 70%

Exam

It contains 40x questions. Please answer the questions on separate paper or in an Excel file, and then compare your answers with those in the answers section after the quiz.

Q1:

What is the output of the following code?

```
import math
print(math.floor(7.8))
```

Options:

- 8
- 7
- 7.8
- It will raise a TypeError

Q2:

Which of the following import statements correctly imports only the `sqrt` function from the `math` module?

```
# your code
```

Options:

- `import math.sqrt`
- `from math import sqrt`
- `import sqrt from math`
- `from math import *`

Q3:

What will be the value of `sys.path` after the following code is executed?

```
import sys
```

```
print(sys.path)
```

Options:

- A list containing the current working directory and standard library paths
- A string representing the current working directory
- An integer indicating the number of modules loaded
- A dictionary of system environment variables

?

Q4:

What does the `__init__.py` file signify in a Python package?

```
# Directory structure
# mypackage/
#   └── __init__.py
#   └── module1.py
```

Options:

- It makes Python treat the directory as a package
- It initializes variables for the package
- It is the main executable file of the package
- It is used to import all modules in the package automatically

?

Q5:

Given the following code, what will be the output?

```
from random import choice
elements = ['apple', 'banana', 'cherry']
print(choice(elements))
```

Options:

- It will print the first element 'apple'
- It will print a random element from the list
- It will print the entire list
- It will raise an AttributeError

?

Q6:

How can you make a variable private in a Python module?

```
# mymodule.py
_var = 10
```

Options:

- Prefix the variable name with a single underscore
- Prefix the variable name with two underscores
- Suffix the variable name with an underscore
- Enclose the variable name in double quotes

?

Q7:

What will be the output of the following code?

```
try:
    x = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero")
else:
    print("Division successful")
finally:
    print("Execution completed")
```

Options:

- "Division successful" followed by "Execution completed"
- "Cannot divide by zero" followed by "Execution completed"
- "Execution completed" only
- An unhandled exception is raised

?

Q8:

How do you define a custom exception in Python?

```
# your code
```

Options:

- class MyException: pass
- def MyException(Exception): pass
- class MyException(Exception): pass
- exception MyException: pass

?

Q9:

What is the purpose of the `assert` statement in Python?

Options:

- To handle exceptions
- To define a function
- To set a breakpoint for debugging
- To test if a condition is True, and raise an `AssertionError` if not

❓ Q10:

What will the following code output?

```
try:  
    x = int("abc")  
except ValueError as e:  
    print(e.args)
```

Options:

- "abc"
- ('invalid literal for int() with base 10: 'abc',)
- An empty tuple
- It will raise a `TypeError`

❓ Q11:

Which block is executed if no exception is raised in a try-except statement?

```
try:  
    # code that may not raise an exception  
except Exception:  
    # handle exception  
else:  
    # what?
```

Options:

- The except block
- The else block
- The try block again
- The finally block

❓ Q12:

What is the output of the following code?

```
s = "Hello, World!"  
print(s[7:12])
```

Options:

- "World"
- "World!"
- "Worl"
- "Hello"

❓ Q13:

Which method would you use to check if a string contains only digits?

```
s = "12345"
```

Options:

- s.isdigit()
- s.isnumeric()
- s.isdecimal()
- All of the above

❓ Q14:

What does the `ord()` function do?

Options:

- Returns the ASCII value of a character
- Converts an integer to a character
- Returns the ordinal position of a character in a string
- Orders a list of characters

❓ Q15:

What will be the output of the following code?

```
s = "Python"  
print(s * 3)
```

Options:

- "PythonPythonPython"
- "Python 3"

- "Python"
- It will raise a TypeError

❓ Q16:

How can you convert the string "hello world" to uppercase?

```
s = "hello world"
```

Options:

- s.uppercase()
- s.toUpperCase()
- s.upper()
- s.capitalize()

❓ Q17:

What is the result of the following code?

```
s = "OpenAI"
print("AI" in s)
```

Options:

- True
- False
- It will raise an exception
- "AI"

❓ Q18:

Which of the following methods splits a string into a list where each word is a list item?

```
s = "Split this string"
```

Options:

- s.split()
- s.divide()
- s.separate()
- s.break()

❓ Q19:

What is the output of the following code?

```
class MyClass:  
    def __init__(self, value):  
        self.value = value  
obj = MyClass(10)  
print(obj.__dict__)
```

Options:

- {'value': 10}
- {'self': }
- {'**init- It will raise an AttributeError**

❓ Q20:

How do you check if an object **obj** is an instance of class **MyClass**?

```
# your code
```

Options:

- isinstance(obj, MyClass)
- type(obj) == MyClass
- obj instanceof MyClass
- obj.isinstance(MyClass)

❓ Q21:

What is polymorphism in Python?

Options:

- A way to encapsulate data
- The ability to define multiple methods with the same name
- The ability of different objects to be accessed through the same interface
- Inheritance from multiple superclasses

❓ Q22:

What does the **self** keyword represent in a class method?

```
class MyClass:  
    def method(self):  
        pass
```

Options:

- The class itself
- An instance of the class
- A global variable
- Nothing special, just a regular variable

❓ Q23:

What is the purpose of the `__str__` method in a Python class?

```
class MyClass:  
    def __str__(self):  
        return "MyClass instance"
```

Options:

- To initialize the class
- To represent the object as a string
- To compare two objects
- To delete the object

❓ Q24:

What will be the output of the following code?

```
class Parent:  
    def greet(self):  
        print("Hello from Parent")  
class Child(Parent):  
    def greet(self):  
        print("Hello from Child")  
obj = Child()  
obj.greet()
```

Options:

- "Hello from Parent"
- "Hello from Child"
- It will raise an AttributeError
- Both messages will be printed

?

Q25:

How can you access the class variables from an instance?

```
class MyClass:  
    class_var = 5  
    def __init__(self):  
        self.instance_var = 10  
obj = MyClass()
```

Options:

- obj.class_var
- obj.instance_var
- MyClass.instance_var
- obj.get_class_var()

?

Q26:

What is inheritance in Python?

Options:

- A way to create a copy of an object
- A mechanism where a new class derives attributes and methods from an existing class
- A method to hide variables
- A way to handle exceptions

?

Q27:

What is the purpose of the __init__ method in a class?

```
class MyClass:  
    def __init__(self, value):  
        self.value = value
```

Options:

- To destroy the object
- To initialize the object's attributes
- To represent the object as a string
- To define class variables

?

Q28:

What will `isinstance(obj, object)` return for any object `obj`?

Options:

- True
- False
- It depends on the object's type
- It will raise a TypeError

❓ Q29:

What does `__name__` represent in a Python module?

Options:

- The name of the function
- The name of the module
- The name of the class
- It is not a special attribute

❓ Q30:

How can you prevent a class from being subclassed?

Options:

- Use a final keyword
- Define **new** to raise an exception
- Python does not support preventing subclassing
- Use a metaclass that restricts subclassing

❓ Q31:

What is the output of the following code?

```
lst = [x**2 for x in range(5) if x % 2 == 0]
print(lst)
```

Options:

- [0, 1, 4, 9, 16]
- [0, 4, 16]
- [0, 4, 8, 12, 16]
- [1, 9]

❓ Q32:

What does the `map()` function do?

```
def square(x):
    return x * x
numbers = [1, 2, 3, 4]
```

```
result = map(square, numbers)
print(list(result))
```

Options:

- It filters the list based on the function
- It applies the function to each item and returns a map object
- It combines two lists into a dictionary
- It sorts the list in ascending order

❓ Q33:

What is the purpose of the `lambda` keyword in Python?

Options:

- To define a regular function
- To create an anonymous function
- To import modules
- To handle exceptions

❓ Q34:

What will the following code output?

```
f = lambda x, y: x + y
print(f(2, 3))
```

Options:

- 23
- 5
- '23'
- It will raise a SyntaxError

❓ Q35:

How does the `filter()` function work in Python?

```
def is_even(x):
    return x % 2 == 0
numbers = [1, 2, 3, 4, 5, 6]
result = filter(is_even, numbers)
print(list(result))
```

Options:

- It maps the function to each item
- It filters items where the function returns True
- It sorts the items
- It reduces the list to a single value

❓ Q36:

What does the `zip()` function do?

```
list1 = [1, 2, 3]
list2 = ['a', 'b', 'c']
zipped = zip(list1, list2)
print(list(zipped))
```

Options:

- Combines the lists into a dictionary
- Creates pairs of elements from both lists
- Appends list2 to list1
- It does nothing, zip is not a function

❓ Q37:

What mode should you use with `open()` to read a binary file?

Options:

- 'r'
- 'w'
- 'rb'
- 'rt'

❓ Q38:

How do you write a file using a context manager in Python?

Options:

- .

```
f = open('file.txt', 'w')
f.write('Hello')
f.close()
```

- .

```
with open('file.txt', 'w') as f:
```

```
f.write('Hello')
```

- write('Hello')
- It cannot be done

?

Q39:

What is the result of the following code?

```
def add(x, y):  
    return x + y  
result = add(*[1, 2])  
print(result)
```

Options:

- [1, 2]
- (1, 2)
- 3
- It will raise a TypeError

?

Q40:

What is a list comprehension in Python?

Options:

- A way to create lists using loops and conditional statements in a single line
- A built-in function to concatenate lists
- A method to reverse a list
- A function to sort a list

✓ The Answers

Question Nr.	Answer	Explanation
Q1	B	<code>math.floor(7.8)</code> returns the largest integer less than or equal to 7.8, which is 7.
Q2	B	<code>from math import sqrt</code> correctly imports only the <code>sqrt</code> function from the <code>math</code> module.
Q3	A	<code>sys.path</code> is a list containing the current working directory and standard library paths.
Q4	A	<code>__init__.py</code> makes Python treat the directory as a package.
Q5	B	<code>choice(elements)</code> selects and prints a random element from the list.
Q6	A	Prefixing the variable name with a single underscore makes it a private variable by convention.
Q7	B	The <code>ZeroDivisionError</code> is caught, printing "Cannot divide by zero" and "Execution completed".
Q8	C	To define a custom exception, subclass from <code>Exception: class MyException(Exception): pass</code>
Q9	D	<code>assert</code> tests if a condition is True, raising <code>AssertionError</code> if not.
Q10	B	<code>e.args</code> contains the error message: (<code>'invalid literal for int() with base 10: 'abc''</code>)
Q11	B	The <code>else</code> block is executed if no exception is raised.
Q12	A	<code>s[7:12]</code> slices the string to "World".
Q13	D	<code>s.isdigit()</code> , <code>s.isnumeric()</code> , and <code>s.isdecimal()</code> all check if the string contains only digits.
Q14	A	<code>ord()</code> returns the ASCII value of a character.
Q15	A	<code>s * 3</code> results in "PythonPythonPython".
Q16	C	<code>s.upper()</code> converts the string to uppercase.
Q17	A	"AI" is a substring of "OpenAI", so <code>in</code> returns True.
Q18	A	<code>s.split()</code> splits the string into a list of words.
Q19	A	<code>obj.__dict__</code> contains <code>{'value': 10}</code> .
Q20	A	<code>isinstance(obj, MyClass)</code> checks if <code>obj</code> is an instance of <code>MyClass</code> .
Q21	C	Polymorphism allows different objects to be accessed through the same interface.
Q22	B	<code>self</code> refers to the instance of the class.

Question Nr.	Answer	Explanation
Q23	B	<code>__str__</code> defines the string representation of an object.
Q24	B	<code>Child.greet()</code> overrides <code>Parent.greet()</code> , so "Hello from Child" is printed.
Q25	A	<code>obj.class_var</code> accesses the class variable from an instance.
Q26	B	Inheritance allows a new class to derive from an existing class.
Q27	B	<code>__init__</code> initializes the object's attributes.
Q28	A	Any object is an instance of <code>object</code> , so it returns True.
Q29	B	<code>__name__</code> represents the name of the module.
Q30	C	Python does not support preventing subclassing directly.
Q31	B	The list comprehension creates [0, 4, 16] by squaring even numbers in range(5).
Q32	B	<code>map()</code> applies the function to each item and returns a map object.
Q33	B	<code>lambda</code> creates an anonymous function.
Q34	B	The lambda function adds 2 and 3, resulting in 5.
Q35	B	<code>filter()</code> filters items where the function returns True.
Q36	B	<code>zip()</code> creates pairs of elements from both lists.
Q37	C	'rb' mode is used to read a binary file.
Q38	B	Using <code>with open() as f:</code> ensures the file is properly closed.
Q39	C	<code>add(*[1, 2])</code> unpacks the list and returns 3.
Q40	A	List comprehension creates lists using loops and conditionals in one line.

Licensed for: WBS Training AG
PYDASC71e-13



4.4 Exam 4

- **Exam Name:** PCAP™ – Certified Associate Python Programmer - **PCAP-31-03**
- **Duration:** 65 minutes
- **Passing Score:** 70%

Exam

It contains 40x questions. Please answer the questions on separate paper or in an Excel file, and then compare your answers with those in the answers section after the quiz.

Q1:

What will be the output of the following code?

```
import math
print(math.ceil(4.2))
```

Options:

- 4
- 4.2
- 5
- It will raise an AttributeError

Q2:

Which of the following statements correctly imports the `random` module with an alias `rnd`?

```
# your code
```

Options:

- import random as rnd
- from random import as rnd
- import rnd from random
- from random import random as rnd

Q3:

What is the purpose of the `dir()` function in Python?

Options:

- To list all files in the current directory
- To list all attributes and methods of an object
- To change the current working directory
- To delete an object

❓ Q4:

Given the following code, what will be the value of `math.__name__`?

```
import math
print(math.__name__)
```

Options:

- "math"
- "math"
- It will raise an AttributeError
- The name of the current module

❓ Q5:

Which of the following correctly adds a directory to `sys.path`?

```
import sys
```

Options:

- `sys.path.append('/new/path')`
- `sys.add_path('/new/path')`
- `sys.insert('/new/path')`
- `sys += '/new/path'`

❓ Q6:

What is the output of the following code?

```
import random
random.seed(10)
print(random.randint(1, 100))
```

Options:

- It will always print the same number
- It will print a different random number each time
- It will raise a ValueError

- It will print 10

❓ Q7:

Which exception is raised when a function receives an argument of the correct type but an inappropriate value?

Options:

- TypeError
- ValueError
- AttributeError
- IndexError

❓ Q8:

What will the following code output?

```
try:  
    assert 2 + 2 == 5, "Math is broken!"  
except AssertionError as e:  
    print(e)
```

Options:

- Nothing, as the assertion passes
- "Math is broken!"
- It will raise a SyntaxError
- "AssertionError"

❓ Q9:

How can you handle multiple specific exceptions in a single except block?

Options:

- Using multiple except statements
- Using a tuple of exceptions in a single except statement
- It's not possible to handle multiple exceptions in one block
- Using a list of exceptions in an except statement

❓ Q10:

What will be the output of the following code?

```
class MyException(Exception):  
    pass  
  
try:
```

```
raise MyException("Custom error message")
except MyException as e:
    print(e)
```

Options:

- It will print nothing
- "Custom error message"
- It will raise a TypeError
- "MyException"

❓ Q11:

Which string method would you use to check if a string starts with a specific substring?

```
s = "GraduateAI"
```

Options:

- s.startswith("Grad")
- s.startsWith("Grad")
- s.start("Grad")
- s.isstart("Grad")

❓ Q12:

What is the result of the following code?

```
s = "DataScience"
print(s[-4:])
```

Options:

- "Data"
- "ence"
- "Sci"
- "nce"

❓ Q13:

Which of the following will convert the string "123" to the integer 123?

```
s = "123"
```

Options:

- int(s)
- str(s)
- float(s)
- bool(s)

?

Q14:

What does the `chr()` function do in Python?

Options:

- Converts a character to its ASCII integer
- Converts an ASCII integer to its corresponding character
- Returns the Unicode code point of a character
- It sorts characters in a string

?

Q15:

What will the following code output?

```
s = "hello"  
print(s.capitalize())
```

Options:

- "Hello"
- "HELLO"
- "hello"
- "hELLO"

?

Q16:

Which method would you use to join a list of strings into a single string with commas?

```
lst = ["apple", "banana", "cherry"]
```

Options:

- lst.join(",")
- ",".join(lst)
- lst.concatenate(",")
- ",".concatenate(lst)

?

Q17:

What is the output of the following code?

```
s = "OpenAI"  
print(s.isupper())
```

Options:

- True
- False
- It will raise an AttributeError
- It depends on the environment

❓ Q18:

How can you reverse a string `s` using slicing?

```
s = "Python"
```

Options:

- `s[::-1]`
- `s[:1]`
- `s[:-2]`
- `s[1:-1]`

❓ Q19:

What will be the output of the following code?

```
class Animal:  
    def __init__(self, name):  
        self.name = name  
  
class Dog(Animal):  
    pass  
  
dog = Dog("Buddy")  
print(dog.name)
```

Options:

- "Dog"
- "Animal"
- "Buddy"
- It will raise an AttributeError

?

Q20:

Which magic method is used to define the behavior of the `+` operator for a class?

```
class Number:  
    def __init__(self, value):  
        self.value = value
```

Options:

- **add**
- **sum**
- **plus**
- **operator**

?

Q21:

What will the following code output?

```
class Base:  
    def __init__(self):  
        self.x = 5  
  
class Derived(Base):  
    def __init__(self):  
        super().__init__()  
        self.y = 10  
  
obj = Derived()  
print(obj.__dict__)
```

Options:

- `{'x': 5}`
- `{'y': 10}`
- `{'x': 5, 'y': 10}`
- It will raise an `AttributeError`

?

Q22:

How do you define a class method in Python?

```
class MyClass:  
    @classmethod  
    def my_method(cls):  
        pass
```

Options:

- Using the `@staticmethod` decorator
- Using the `@classmethod` decorator
- Using the `@classmethod` keyword in the method signature
- Using the `def classmethod()` syntax

?

Q23:

What does the `isinstance()` function return when checking for inheritance?

```
class A:  
    pass  
  
class B(A):  
    pass  
  
b = B()  
print(isinstance(b, A))
```

Options:

- True
- False
- It will raise a `TypeError`
- It depends on the implementation

?

Q24:

What is the purpose of the `__del__` method in a Python class?

```
class MyClass:  
    def __del__(self):  
        print("Destructor called")
```

Options:

- To initialize the object
- To represent the object as a string
- To clean up resources when the object is destroyed
- To compare two objects

?

Q25:

Which of the following statements is true about private variables in Python?

Options:

- They cannot be accessed outside the class
- They are enforced by the Python interpreter
- They are a convention to indicate intended privacy
- They are defined using a single underscore prefix

❓ Q26:

What will be the output of the following code?

```
class Rectangle:  
    def __init__(self, width, height):  
        self.width = width  
        self.height = height  
  
    def area(self):  
        return self.width * self.height  
  
rect = Rectangle(3, 4)  
print(rect.area())
```

Options:

- 7
- 12
- 34
- It will raise a TypeError

❓ Q27:

How can you create a destructor in a Python class?

```
class MyClass:  
    def __init__(self):  
        pass
```

Options:

- Define a method named `__del__`
- Define a method named `destroy`
- Use the `@destructor` decorator
- It is not possible to create a destructor in Python

❓ Q28:

What is the output of the following code?

```
class A:  
    pass  
  
print(isinstance(A(), object))
```

Options:

- True
- False
- It will raise a TypeError
- It depends on the class definition

❓ Q29:

Which of the following is a correct way to define multiple inheritance in Python?

```
class A:  
    pass  
  
class B:  
    pass  
  
class C(A, B):  
    pass
```

Options:

- class C(A & B):
- class C(A | B):
- class C(A, B):
- class C(A + B):

❓ Q30:

What will be the output of the following code?

```
class MyClass:  
    def __init__(self):  
        self.__private = "secret"  
  
obj = MyClass()  
print(obj.__private)
```

Options:

- "secret"

- It will raise an AttributeError
- "__private"
- None

❓ Q31:

What is a closure in Python?

Options:

- A function that is closed and cannot be modified
- A nested function that remembers the environment in which it was created
- A class with private methods
- A function that does not return anything

❓ Q32:

What will the following code output?

```
def outer(x):
    def inner(y):
        return x + y
    return inner

add_five = outer(5)
print(add_five(10))
```

Options:

- 5
- 10
- 15
- It will raise a TypeError

❓ Q33:

Which of the following is a valid list comprehension to create a list of squares of even numbers from 0 to 9?

```
# your code
```

Options:

- [x**2 for x in range(10) if x % 2 == 0]
- [x^2 for x in range(10) if x % 2]
- [x**2 if x % 2 == 0 for x in range(10)]
- [x**2 for x in range(10) else x % 2 == 0]

?

Q34:

What does the following lambda function do?

```
f = lambda x: x * 2
print(f(4))
```

Options:

- It returns 8
- It returns 4
- It returns "x * 2"
- It raises a SyntaxError

?

Q35:

How can you apply a function to all items in a list using `map()`?

```
def square(x):
    return x * x

numbers = [1, 2, 3, 4]
```

Options:

- `map(square, numbers)`
- `map(numbers, square)`
- `numbers.map(square)`
- `square.map(numbers)`

?

Q36:

What will be the output of the following code?

```
def is_positive(x):
    return x > 0

numbers = [-2, -1, 0, 1, 2]
filtered = filter(is_positive, numbers)
print(list(filtered))
```

Options:

- `[-2, -1, 0, 1, 2]`
- `[1, 2]`
- `[0, 1, 2]`

- It will raise a TypeError

❓ Q37:

What does the `open()` function return when used to open a file?

```
f = open('example.txt', 'r')
```

Options:

- A file object
- A string containing the file contents
- A list of lines in the file
- It does not return anything

❓ Q38:

Which mode should you use with `open()` to append to a text file?

Options:

- 'a'
- 'w'
- 'r'
- 'x'

❓ Q39:

What is the result of the following code?

```
data = b'Hello'  
print(data.decode('utf-8'))
```

Options:

- b'Hello'
- "Hello"
- It will raise a UnicodeDecodeError
- [72, 101, 108, 108, 111]

❓ Q40:

What is the output of the following code?

```
with open('test.bin', 'wb') as f:  
    f.write(b'\x00\xFF')
```

```
with open('test.bin', 'rb') as f:  
    print(f.read())
```

Options:

- b'\x00\xFF'
 - "00FF"
 - It will raise an IOError
 - [0, 255]
-

Licensed for: WBS Training AG
PYDASC71e-13

✓ The Answers

Question Nr.	Answer	Explanation
Q1	C	<code>math.ceil(4.2)</code> returns the smallest integer greater than or equal to 4.2, which is 5.
Q2	A	<code>import random as rnd</code> correctly imports the <code>random</code> module with the alias <code>rnd</code> .
Q3	B	<code>dir()</code> lists all attributes and methods of an object.
Q4	A	<code>math.__name__</code> returns the name of the module, which is "math".
Q5	A	<code>sys.path.append('/new/path')</code> correctly adds a new path to <code>sys.path</code> .
Q6	A	Seeding with <code>random.seed(10)</code> makes the output of <code>randint</code> deterministic, always the same number.
Q7	B	<code>ValueError</code> is raised when a function receives an argument of correct type but inappropriate value.
Q8	B	The assertion fails, raising an <code>AssertionError</code> with the message "Math is broken!".
Q9	B	Multiple exceptions can be handled using a tuple in a single <code>except</code> statement, e.g., <code>except (TypeError, ValueError):</code>
Q10	B	Raising <code>MyException("Custom error message")</code> and catching it prints "Custom error message".
Q11	A	<code>s.startswith("Grad")</code> checks if the string starts with "Grad".
Q12	B	<code>s[-4:]</code> slices the last four characters, resulting in "ence".
Q13	A	<code>int(s)</code> converts the string "123" to the integer 123.
Q14	B	<code>chr()</code> converts an ASCII integer to its corresponding character.
Q15	A	<code>s.capitalize()</code> capitalizes the first character, resulting in "Hello".
Q16	B	<code>",".join(lst)</code> joins the list elements with commas, resulting in "apple,banana,cherry".
Q17	B	"hello".isupper() returns <code>False</code> as not all characters are uppercase.
Q18	A	<code>s[::-1]</code> reverses the string, resulting in "nohtyP".
Q19	C	The <code>Dog</code> class inherits <code>name</code> from <code>Animal</code> , so <code>dog.name</code> is "Buddy".
Q20	A	The <code>__add__</code> magic method defines the behavior of the <code>+</code> operator.
Q21	C	<code>obj.__dict__</code> contains both <code>x</code> and <code>y</code> with their values: <code>{'x': 5, 'y': 10}</code> .
Q22	B	The <code>@classmethod</code> decorator defines a class method.
Q23	A	<code>isinstance(b, A)</code> returns <code>True</code> because <code>B</code> inherits from <code>A</code> .

Question Nr.	Answer	Explanation
Q24	C	The <code>__del__</code> method is called when an object is about to be destroyed, allowing cleanup.
Q25	C	Private variables in Python are a convention to indicate intended privacy, not enforced by the interpreter.
Q26	B	The <code>area</code> method returns <code>3 * 4 = 12</code> .
Q27	A	Defining a method named <code>__del__</code> creates a destructor.
Q28	A	All objects in Python are instances of <code>object</code> , so it returns <code>True</code> .
Q29	C	<code>class C(A, B):</code> correctly defines multiple inheritance from classes <code>A</code> and <code>B</code> .
Q30	B	Accessing <code>obj.__private</code> raises an <code>AttributeError</code> due to name mangling.
Q31	B	A closure is a nested function that remembers its enclosing environment.
Q32	C	<code>add_five(10)</code> returns <code>5 + 10 = 15</code> .
Q33	A	<code>[x**2 for x in range(10) if x % 2 == 0]</code> creates a list of squares of even numbers from 0 to 9.
Q34	A	The lambda function doubles the input, so <code>f(4)</code> returns <code>8</code> .
Q35	A	<code>map(square, numbers)</code> applies <code>square</code> to each item in <code>numbers</code> .
Q36	B	<code>filter(is_positive, numbers)</code> filters and returns <code>[1, 2]</code> .
Q37	A	<code>open()</code> returns a file object.
Q38	A	'a' mode opens the file for appending in text mode.
Q39	B	<code>data.decode('utf-8')</code> converts the byte string to "Hello".
Q40	A	Writing and reading in binary mode returns <code>b'\x00\xFF'</code> .