VedaIT

Python Basics Question Paper

Introduction to Python:

1. What is Python, and why is it popular for programming?

- Answer: Python is a high-level, interpreted programming language known for its simplicity and readability. It is popular for its wide range of applications, including web development, data analysis, artificial intelligence, and more.

2. Describe some key features of Python as a programming language.

- Answer: Key features of Python include readability, a rich standard library, dynamic typing, automatic memory management, and support for multiple programming paradigms (procedural, object-oriented, functional).

3. What is the Python Software Foundation (PSF), and what role does it play in the Python community?

- Answer: The Python Software Foundation (PSF) is a nonprofit organization that manages and promotes Python. It plays a key role in supporting the development of Python, organizing events like PyCon, and overseeing the Python community's activities.

4. How do you write a comment in Python code?

- Answer: In Python, comments are written using the `#` symbol. Comments are ignored by the Python interpreter and are used to provide explanations or documentation within the code.

Data Types:

1. Name a few built-in data types in Python.
- Answer: Built-in data types in Python include integers, floating-point numbers, strings, lists, tuples, dictionaries, sets, and more.

2. Explain the difference between integers and floating-point numbers in Python.
- Answer: Integers represent whole numbers, while floating-point numbers represent numbers with a decimal point.

3. What is a string in Python? Provide an example of a string.

- Answer: A string is a sequence of characters enclosed in single, double, or triple quotes. Example: `"Hello, World!"`

4. How do you check the data type of a variable in Python?

- Answer: You can check the data type of a variable using the `type()` function. For example, `type(42)` returns `<class 'int'>`.

5. What is type casting, and how is it done in Python?

- Answer: Type casting, also known as type conversion, is the process of converting a value from one data type to another. It is done using constructors or conversion functions, such as `int()`, `float()`, and `str()`.

If Statements:

1. What is the purpose of an "if" statement in Python?

- Answer: An "if" statement in Python is used for conditional execution. It allows you to execute a block of code only if a specified condition is true.

2. How do you write an "if" statement with a single condition?

- Answer: An "if" statement with a single condition is written as follows:
- python

```
if condition:
    # Code to execute if the condition is true
```

1. What is an "else" statement, and when is it used with "if" statements?

Answer: An "else" statement is used to provide an alternative code block to execute if the "if" condition is false.

2. Explain the concept of "elif" in Python and provide an example.

Answer: "elif" is short for "else if" and is used to specify additional conditions to check if the initial "if" condition is false. It allows for multiple conditional branches in code.

3. How do you write a nested "if" statement?

Answer: A nested "if" statement is an "if" statement inside another "if" or "elif" block. It is used for more complex conditional logic.

While Loops:

1. What is the primary use of a "while" loop in Python?

Answer: A "while" loop in Python is used for repeated execution of a block of code as long as a specified condition is true.

2. Write a "while" loop that counts from 1 to 5 and prints the numbers.

Answer:

```
num = 1
while num <= 5:
    print(num)
    num += 1
```

3. What is an infinite loop, and how can you avoid it when using "while" loops?

Answer: An infinite loop is a loop that runs indefinitely because its termination condition is never met. To avoid it, ensure that the condition in a "while" loop will eventually become false.

4. How do you exit a "while" loop prematurely using a "break" statement?

Answer: You can exit a "while" loop prematurely using the "break" statement. It immediately exits the loop and continues with the next statement after the loop.

For Loops:

1. When is a "for" loop typically used in Python?

Answer: A "for" loop is typically used in Python for iterating over a sequence (such as a list, tuple, or string) or for repeating a block of code a specific number of times.

2. Write a Python "for" loop to iterate over a list of names and print each name.

```
names = ["Alice", "Bob", "Charlie"]
for name in names:
    print(name)
```

3. Explain the purpose of the "range()" function in "for" loops.

Answer: The "range()" function generates a sequence of numbers that can be used in "for" loops. For example, `range(5)` generates `[0, 1, 2, 3, 4]`.

4. How can you iterate over the keys of a dictionary using a "for" loop?

Answer: To iterate over the keys of a dictionary using a "for" loop, you can use the `.keys()` method or simply loop directly through the dictionary.

5. What is the main difference between "for" loops and "while" loops in Python?

Answer: The main difference between "for" loops and "while" loops is that "for" loops are typically used when you know the number of iterations in advance and want to iterate over a sequence, while "while" loops are used when you want to repeat a block of code as long as a condition is true.

Lists and Tuples:

1. What is the main difference between a list and a tuple in Python?

Answer: The main difference is that lists are mutable (modifiable), while tuples are immutable (unchangeable). In other words, you can add, remove, or modify elements in a list after it's created, but you cannot do the same with a tuple.

2. How do you add an element to the end of a list?

Answer: You can add an element to the end of a list using the `append()` **method. For example:**

my_list = [1, 2, 3]
my_list.append(4)
# Now, my_list contains [1, 2, 3, 4]

3. Can you change the elements of a tuple after it's created? Why or why not?

Answer: No, you cannot change the elements of a tuple after it's created. Tuples are immutable, which means their elements cannot be modified, added, or removed once the tuple is defined. This property ensures data integrity and can be useful for cases where you want to ensure that the data remains constant.

4. Explain the difference between indexing and slicing in lists.

Answer: Indexing is the process of accessing a specific element in a list by its position. In Python, indexing starts from 0. For example, to access the first element of a list `my_list`**, you use** `my_list[0]`**.**

Slicing, on the other hand, is used to extract a portion (sublist) of a list by specifying a range of indices. It is done using the colon `:` **operator. For example,** `my_list[1:4]` **would extract elements at indices 1, 2, and 3 (not including 4).**

5. Write Python code to concatenate two lists.

Answer:

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
concatenated_list = list1 + list2
# The concatenated_list will be [1, 2, 3, 4, 5, 6]
```

6. How can you find the length of a list or tuple in Python?

Answer: You can find the length of a list or tuple using the built-in `len()` **function. For example:**

```
my_list = [1, 2, 3, 4, 5]
length_of_list = len(my_list)  # This will be 5
```

Dictionaries

1. What is a key-value pair in a dictionary?

Answer: A key-value pair in a dictionary is a combination of two elements: a key and a corresponding value. The key is used to uniquely identify the value, and the value can be any Python data type (e.g., integers, strings, lists, or even other dictionaries).

2. How do you access the value associated with a specific key in a dictionary?

Answer: You can access the value associated with a specific key in a dictionary by using the key inside square brackets `[]` **or by using the** `.get()` **method. For example:**

```
my_dict = {'name': 'Alice', 'age': 30}
name = my_dict['name']  # Accessing value using square brackets
age = my_dict.get('age')  # Accessing value using .get() method
```

3. Can a dictionary have multiple keys with the same value?

Answer: Yes, a dictionary can have multiple keys with the same value. However, each key in a dictionary must be unique. Multiple keys can map to the same value, but the values themselves do not need to be unique.

4. How do you add a new key-value pair to an existing dictionary?

Answer: You can add a new key-value pair to an existing dictionary by simply assigning a value to a new key. For example:

```
my_dict = {'name': 'Alice', 'age': 30}
my_dict['city'] = 'New York'  # Adding a new key-value pair
```

5. Write Python code to iterate over the keys of a dictionary and print their values.

Answer:

```
my_dict = {'name': 'Alice', 'age': 30, 'city': 'New York'}

# Iterating over keys and printing their values
for key in my_dict:
   value = my_dict[key]
   print(f"Key: {key}, Value: {value}")
```

String Functions:

1. Explain the purpose of the split() function in Python and provide an example.

Answer: The `split()` **function in Python is used to split a string into a list of substrings based on a specified delimiter (separator). It is commonly used for parsing text data.**
Example:

```
text = "Hello, World! This is a sample sentence."
words = text.split()  # Split the text using the default delimiter (space)
print(words)
```

Output

```
['Hello,', 'World!', 'This', 'is', 'a', 'sample', 'sentence.']
```

2. What is the difference between the strip(), lstrip(), and rstrip() methods for strings?

Answer: These methods are used to remove whitespace (or other specified characters) from the beginning and/or end of a string:

- `strip()`: **Removes whitespace (or specified characters) from both the beginning and end of the string.**
- `lstrip()`: **Removes whitespace (or specified characters) from the beginning (left) of the string.**
- `rstrip()`: **Removes whitespace (or specified characters) from the end (right) of the string.**

3. How can you check if a string contains a specific substring?

Answer: You can check if a string contains a specific substring using the `in` operator or the `str.find()` method. Here are examples using both approaches:

- **Using `in` operator:**

```
text = "Hello, World!"
substring = "World"
if substring in text:
   print("Substring found!")
else:
   print("Substring not found.")
```

Using `str.find()` method:

```
text = "Hello, World!"
substring = "World"
if text.find(substring) != -1:
   print("Substring found!")
else:
   print("Substring not found.")
```

4. Write code to convert a string to lowercase and then to uppercase.

Answer:

```
text = "Hello, World!"
```

```
lowercase_text = text.lower()  # Convert to lowercase
uppercase_text = text.upper()  # Convert to uppercase

print("Lowercase:", lowercase_text)
print("Uppercase:", uppercase_text)
```

Output:

```
Lowercase: hello, world!
Uppercase: HELLO, WORLD!
```

If Statements and Nested If Statements:

1. What is the purpose of an if statement in Python?
   Answer: The purpose of an "if" statement in Python is to execute a block of code conditionally. It allows you to perform actions based on whether a specified condition is true or false.
2. How do you write an if-else statement in Python?
   Answer: An if-else statement in Python is written using the `if` **keyword to specify the condition to check, followed by a code block to execute if the condition is true. Optionally, you can include an** `else` **block for code to execute when the condition is false.**

```
if condition:
   # Code to execute if condition is true
else:
   # Code to execute if condition is false
```

3. Explain the concept of nested if statements. Provide an example.
Answer: Nested if statements are if statements placed inside another if or elif block. They allow for more complex conditional logic. The inner if statement is only evaluated if the outer if or elif condition is true.

```
x = 10
if x > 5:
   if x < 15:
      print("x is between 5 and 15")
```

4. Write Python code to check if a number is positive, negative, or zero using if-elif-else.

Answer:

```
num = float(input("Enter a number: "))

if num > 0:
   print("Positive number")
elif num < 0:
   print("Negative number")
else:
   print("Zero")
```

While Loops and Nested While Loops:

1. What is the purpose of a while loop in Python?
   Answer: The purpose of a "while" loop in Python is to repeatedly execute a block of code as long as a specified condition remains true. It is used for tasks where the number of iterations is not known in advance.
2. **How do you exit a while loop prematurely using a break statement?**
   Answer: You can exit a "while" loop prematurely using the "break" statement. When "break" is encountered within a loop, it immediately exits the loop, and control flows to the code following the loop.
3. **Explain the concept of nested while loops. Provide an example.**
   Answer: Nested while loops are "while" loops placed inside another "while" loop. They allow for the execution of more complex repetitive tasks. The inner "while" loop runs to completion for each iteration of the outer "while" loop.

```
outer_count = 0
while outer_count < 3:
   inner_count = 0
   while inner_count < 2:
      print(f"Outer: {outer_count}, Inner: {inner_count}")
      inner_count += 1
   outer_count += 1
```

4. Write Python code to print all even numbers from 1 to 10 using a while loop.

Answer:

```
num = 1
while num <= 10:
   if num % 2 == 0:
    print(num)
  num += 1
```

5. What is the difference between a for loop and a while loop in Python?

Answer: The main difference is that a "for" loop is typically used when you know the number of iterations in advance (e.g., when iterating over a sequence), whereas a "while" loop is used when you want to repeat a block of code as long as a specified condition remains true. "For" loops are more concise when iterating over sequences, while "while" loops offer more flexibility for arbitrary looping conditions.

Python Programs

Python programs with solutions that cover lists, tuples, dictionaries, if statements, sets, elif, while loops, and nested while loops:

1. Lists:

Program: Find the sum of all elements in a list.

```
my_list = [1, 2, 3, 4, 5]
total = sum(my_list)
print("Sum of elements in the list:", total)
```

2. Tuples:

Program: Access and print elements from a tuple.

```
my_tuple = (10, 20, 30, 40, 50)
print("Third element of the tuple:", my_tuple[2])
```

3. Dictionaries:

Program: Count the frequency of elements in a list using a dictionary.

```
my_list = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]
count_dict = {}
for item in my_list:
    if item in count_dict:
        count_dict[item] += 1
    else:
        count_dict[item] = 1
print("Element frequencies:", count_dict)
```

4. If Statements:

Program: Check if a number is positive, negative, or zero.

```
num = 7
if num > 0:
    print("Positive")
elif num < 0:
    print("Negative")
else:
    print("Zero")
```

5. Sets:

Program: Find the intersection of two sets.

```
set1 = {1, 2, 3, 4, 5}
set2 = {3, 4, 5, 6, 7}
intersection = set1.intersection(set2)
print("Intersection of sets:", intersection)
```

6. Elif Statements:

Program: Determine the grade based on a student's score.

```
score = 85
if score >= 90:
    grade = "A"
elif score >= 80:
    grade = "B"
elif score >= 70:
    grade = "C"
elif score >= 60:
    grade = "D"
else:
    grade = "F"
print("Grade:", grade)
```

7. While Loops:

Program: Print numbers from 1 to 5 using a while loop.

```
num = 1
while num <= 5:
    print(num)
    num += 1
```

8. Nested While Loops:

Program: Print a pattern of asterisks using nested while loops.

```
rows = 5
i = 1
while i <= rows:
    j = 1
    while j <= i:
        print("*", end=' ')
        j += 1
    print()
    i += 1
```

1. Print a Pattern:

Program:

```
rows = 5
for i in range(1, rows + 1):
    for j in range(1, i + 1):
        print("*", end=' ')
    print()
```

Output:

```
*
* *
* * *
* * * *
* * * * *
```

2. Multiplication Table:

Program:

```
for i in range(1, 11):
    for j in range(1, 11):
        print(i * j, end='\t')
    print()
```

Output:

```
1       2       3       4       5       6       7       8       9       10
2       4       6       8       10      12      14      16      18      20
3       6       9       12      15      18      21      24      27      30
4       8       12      16      20      24      28      32      36      40
5       10      15      20      25      30      35      40      45      50
6       12      18      24      30      36      42      48      54      60
7       14      21      28      35      42      49      56      63      70
8       16      24      32      40      48      56      64      72      80
9       18      27      36      45      54      63      72      81      90
10      20      30      40      50      60      70      80      90      100
```

3. Matrix Addition:

Program:

```
matrix1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
matrix2 = [[9, 8, 7], [6, 5, 4], [3, 2, 1]]
result = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]

for i in range(len(matrix1)):
    for j in range(len(matrix1[0])):
        result[i][j] = matrix1[i][j] + matrix2[i][j]

for row in result:
    print(row)
```

Output:

```
[10, 10, 10]
[10, 10, 10]
[10, 10, 10]
```

4. Nested Loop with Strings:

Program:

```
words = ["apple", "banana", "cherry"]
for word in words:
    for letter in word:
        print(letter, end=' ')
    print()
```

Output:

```
a p p l e
b a n a n a
c h e r r y
```

5. Chessboard Pattern:

Program:

```
size = 8
for i in range(size):
    for j in range(size):
        if (i + j) % 2 == 0:
            print("*", end=' ')
        else:
            print(" ", end=' ')
    print()
```

Output:

```
*   *   *   *
  *   *   *   *
*   *   *   *
  *   *   *   *
*   *   *   *
  *   *   *   *
*   *   *   *
  *   *   *   *
```