UNIT 5: STRINGS

1. Introduction to Strings

• What is a String?

A string is a sequence of characters used to represent text.

Definition

A sequence of characters enclosed in quotes.

• Examples of Strings

- Single Quotes ⟶ 'Hello, World!'
- Double Quotes ⟶ "Hello, World!"
- Triple Quotes ⟶ """This is a multi-line string."""

Real-world example

Storing a user's name or an address.

Pro Tip

Use strings to handle text data like names, messages, or addresses in your programs.

Remember:

Strings are Immutable! ☆

Key Takeaway

☆ Strings are fundamental in programming for handling and manipulating text.

Topic : String Basics & Creation

* Creating Strings

In Python, strings can be created by enclosing characters inside single quotes (' '), double quotes (" ") or triple quotes (''' ''' or """ """).

```
① s1 = 'Hello'              # using single quotes
② s2 = "Python"             # using double quotes
③ s3 = '''Multi-line        # using triple quotes
          string              (multi-line string)
          Example'''
```

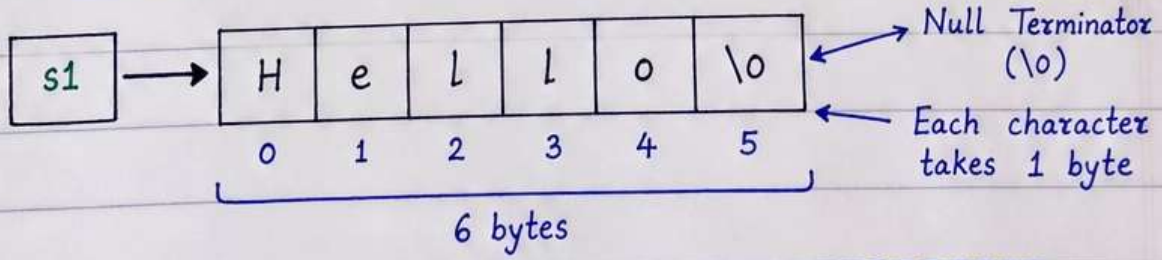
Common Mistake !

Mixing quote types like "Hello is a SyntaxError! ✗

Memory Trick

Quotes are like bread for a sandwich – they must match!

* How String is Stored in Memory ?



Key Points

- Strings are immutable.
- Indexing starts from 0.
- Python stores strings as a sequence of characters.

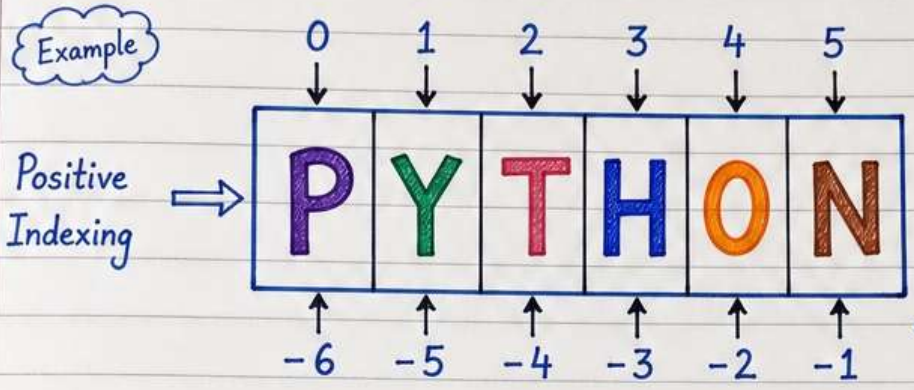
S.P Group of Institute

Date : 18 / 05 / 2024
Page No. : 27

Topic : String Indexing ☆

→ In Python, each character in a string has a unique index (position) which helps us to access that character.

Example



Remember Both are important ! ☺

☆ 1. Positive Indexing (starts from 0)

It starts from the beginning of the string.

- Leftmost character has index 0.

☆ 2. Negative Indexing (starts from -1)

It starts from the end of the string.

- Rightmost character has index -1.

Positive → 0 to n-1
Negative → -1 to -n

Code Examples :

①
```
s = "PYTHON"
print (s[0])   # First character
```

②
```
s = "PYTHON"
print (s[-1])   # Last character
```

Output
P
N

☆ ☆

☆ Quick Revision ☆

	Rule	Positive Indexing	Negative Indexing	Example (s = "PYTHON")
1.	Starts From	0 (Left to Right)	-1 (Right to Left)	s[0] = 'P' , s[-1] = 'N'
2.	Direction	Left → Right	Right → Left	s[1] = 'Y' , s[-2] = 'O'
3.	Range	0 to n-1	-1 to -n	0 to 5 , -1 to -6
4.	Total Characters	n	n	len(s) = 6

☆ Practice more → Understand better → Code like a pro ! ☆

Topic : String Slicing

* Slicing : Cutting the String !

Slicing allows us to extract a part (substring) from a string using the slice operator (colon) :

Syntax : `String [start : stop : step]`

- **start** → The index where slicing begins (inclusive).
  If omitted, it defaults to 0 (start of the string).
- **stop** → The index where slicing ends (exclusive).
  If omitted, it defaults to the length of the string.
- **step** → The number of characters to jump.
  If omitted, it defaults to 1.
  Can be negative (for reverse jumping).

Pro Tip

To reverse a string, use :

`s[::-1]`

Example :
s = "PYTHON"
s[::-1] → "NOHTYP"

Examples :

s = "PYTHON"

	0	1	2	3	4	5
	P	Y	T	H	O	N

① `s[1:4]` → characters from index 1 to 3 (stop-1) → "YTH"

② `s[:3]` → characters from start to index 2 → "PYT"

③ `s[2:]` → characters from index 2 to the end → "THON"

④ `s[::2]` → every 2nd character from start to end → "PTO"

Mini-Exercise

Q. What will `s[1:5:2]` give for "PYTHON"? (Try it yourself !)

(Answer in small text) →

Date : 21 / 05 / 2024
Page No. : 01

S.P Group of Institute

Topic : String Methods - Case Conversion
Language : Python

> ☆ Important
> These methods return a new string. Original string remains unchanged.

String Methods (Part 1)

Methods :

1. upper()
   - Description : Converts all characters to uppercase.

```
s = "Hello World"
s.upper()
```
→ Output: 'HELLO WORLD'

2. Lower()
   - Description : Converts all characters to lowercase.

```
s = "Hello World"
s.lower()
```
→ Output: 'hello world'

3. title()
   - Description : Converts the first character of each word to uppercase, and the rest to lowercase.

```
s = "Hello world from python"
s.title()
```
→ Output: 'Hello World From Python'

4. capitalize()
   - Description : Converts the first character to uppercase, and the rest to lowercase.

```
s = "hello world"
s.capitalize()
```
→ Output: 'Hello world'

5. swapcase()
   - Description : Swaps the case of each character.

```
s = "Hello World 123"
s.swapcase()
```
→ Output: 'hELLO wORLD 123'

Real-world example:
Normalizing user input (e.g., converting email to lowercase).

S.P Group of Institute

Date : 24 / 05 / 2025
Page No. : 31

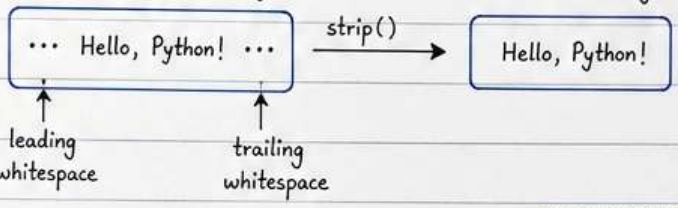
Topic : String Methods - Cleaning & Splitting

String Methods (Part 2)

These methods help in cleaning up strings and working with words.

① strip() - Removes leading and trailing whitespace.

Visual Explanation:



Code Example:

```
s = "   Hello, Python!  \n "
clean = s.strip()
print(clean)
```

Output:

```
Hello, Python!
```

② lstrip() - Removes leading whitespace only.

Code Example:

```
s = "   Hello, Python!  \t "
clean = s.lstrip()
print(clean)
```

Output:

```
Hello, Python!  \t
```

③ rstrip() - Removes trailing whitespace only.

Code Example:

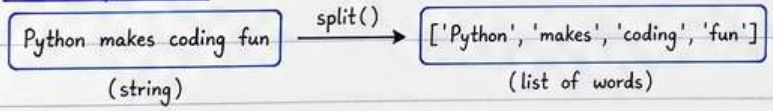
```
s = "   Hello, Python!  \n\n "
clean = s.rstrip()
print(clean)
```

Output:

```
··· Hello, Python!
```

④ split() - Cuts a sentence into a list of words.

Visual Explanation:



Code Example:

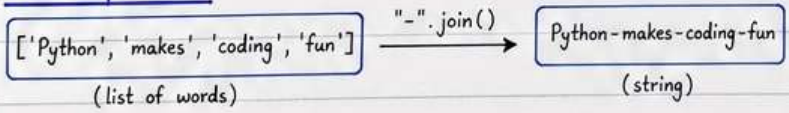
```
s = "Python makes coding fun"
words = s.split()
print(words)
```

Output:

```
['Python', 'makes', 'coding', 'fun']
```

⑤ join() - Glues words together using a separator.

Visual Explanation:



Code Example:

```
words = ['Python', 'makes', 'coding', 'fun']
s = "-".join(words)
print(s)
```

Output:

```
Python-makes-coding-fun
```

☆ Common Mistake

Methods don't change the original string: they return a new one!

(Immutability reminder)

```
s = "  hello  "
s.strip()    # returns 'hello'
print(s)     # still '  hello  ' (unchanged)
```

S.P Group of Institute

Date : 24/05/2024
Page No. : 47

Topic : String Methods - Searching & Checking

String Methods (Part 3)

Commonly Used String Methods:

① find() → Returns the lowest index of the substring if found, else -1
② count() → Returns the number of occurrences of substring in the string
③ startswith() → Returns True if the string starts with the specified prefix
④ endswith() → Returns True if the string ends with the specified suffix
⑤ isalpha() → Returns True if all characters in the string are alphabetic
⑥ isdigit() → Returns True if all characters in the string are digits

find() vs index():

Aspect	find()	index()
Returns	Index of first occurrence	Index of first occurrence
If Not Found	Returns -1	Raises ValueError
Use When	You want safe searching	You want an error if not found
Example	s.find('x')	s.index('x')

Example - Checking if a String is Numeric

```
s1 = "12345"
s2 = "12a45"
print(s1.isdigit())   # True
print(s2.isdigit())   # False
# Another way using all() and isdigit()
def is_numeric(s):
    return len(s) > 0 and all(ch.isdigit() for ch in s)
print(is_numeric(s1))   # True
print(is_numeric(s2))   # False
```

Pro Tip

find() returns -1 if not found, index() raises an error! ☆

More Examples:

```
s = "Hello, Python!"
s.find('o')          # 4
s.count('l')         # 2
s.startswith('Hel')  # True
s.endswith('!')      # True
s.isalpha()          # False
s.isdigit()          # False
```

Practice makes perfect ☺

S.P Group of Institute

Date : 22/05/2024
Page No. : 27

Escape Characters: The Secret Codes!

Escape characters are special codes used inside strings to represent characters that are difficult to type or have a special meaning.

Common Escape Characters:

Escape Sequence	Meaning	Code Example (Inside Quotes)	Before Output	After Output
\n	New Line	"Hello\nWorld"	Hello\nWorld	Hello ↓ World
\t	Tab	"Python\tRocks"	Python\tRocks	Python Rocks
\\	Backslash	"C:\\Users\\name"	C:\\Users\name	C:\Users\name
\'	Single Quote	"It\'s fine"	It\'s fine	It's fine
\"	Double Quote	"She said, \"Hi!\""	She said, \"Hi!\"	She said, "Hi!"

Common Mistake

Forgetting the backslash in file paths like `C:\users\name` ✗

Pro Tip

Use Raw Strings (r'...') for file paths.
It treats backslashes as normal characters.

Example: `r'C:\users\name\documents"` ✓ (Best Practice!)

Think Smart, Code Smarter!

☆ Little codes, big impact! Unlock the power of strings.

String Formatting

* String Formatting: Making it Pretty!

String formatting allows us to insert variables or values inside strings in a readable way.

1) % Operator (Old Style)

```
name = "Aarav"
age = 20
msg = "My name is %s and I am %d years old." % (name, age)
print(msg)
```

Output :

```
My name is Aarav and
I am 20 years old.
```

2) .format() Method

```
name = "Aarav"
age = 20
msg = "My name is {} and I am {} years old.".format(name, age)
print(msg)
```

Output :

```
My name is Aarav and
I am 20 years old.
```

3) f-strings (Modern Style)

```
name = "Aarav"
age = 20
msg = f"My name is {name} and I am {age} years old."
print(msg)
```

Output :

```
My name is Aarav and
I am 20 years old.
```

Pro Tip:

f-strings are the fastest and most readable! ☺

More Examples with f-strings:

```
name = "Aarav"
age = 20
print(f"Hello {name}")                          # Simple variable
print(f"You will be {age + 5} in 5 years.")     # Expression
print(f"PI value is {3.14159:.2f}")             # Formatting numbers
print(f"|{name:<10}|{age:>5}|{3.14159:^10.2f}|") # Alignment (left, right, center) (और भी है)
```

Output :

```
Hello Aarav
You will be 25 in 5 years.
PI value is 3.14
|Aarav     |   20|   3.14   |
```

Comparison Table

Feature	% Operator (Old Style)	.format() Method (Better)	f-strings (Best / Modern)
Syntax	"%s %d" % (a, b)	"{} {}".format(a, b)	f"{a} {b}"
Readability	Low	Medium	High
Performance	Slowest	Faster	Fastest
Python Version	All versions	2.6 +	3.6 +
Best For	Legacy code	Compatibility	New code / Best practice

S.P Group of Institute

Page No. 23
Date : 14 / 05 / 2025

Topic: String Operations & Immutability

* String Operations

1. Concatenation (+)

Combines two or more strings.

```
s1 = "Hello"
s2 = "World"
s3 = s1 + " " + s2
print(s3)
```

Output:

Hello World

2. Repetition (*)

Repeats the string multiple times.

```
s = "Ha"
print(s * 3)
print("Na" * 4)
```

Output:

HaHaHa
NaNaNaNa

3. Membership (in, not in)

Checks if a substring exists in a string.

```
s = "Python Programming"
print("Python" in s)
print("Java" in s)
print("gram" not in s)
```

Output:

True
False
False

Note:

String operations do not modify the original strings (strings are immutable).

Immutability

Strings in Python are immutable, meaning you cannot change, add, or remove characters in an existing string.

Example:

```
s = "APPLE"
s[0] = 'A'  # Trying to change
print(s)
```

Output:

TypeError: 'str' object does not support item assignment

Why?

String object in memory

A	P	P	L	E
0	1	2	3	4

↑
s[0]

We are trying to change 'A' to something else. But strings are immutable, so Python raises TypeError.

Memory Trick

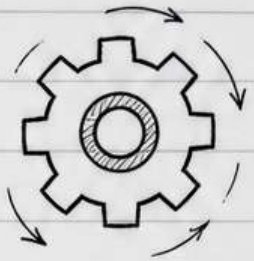
Strings are like statues - once carved, they can't be changed, only replaced!

S.P Group of Institute

Date : 20/05/2025
Page No.: 37

Topic : Intermediate String Concepts - Iteration

Iterating Through Strings



We can iterate through each character in a string in multiple ways.

① Using for loop (for char in s)

```
s = "Python"
for char in s:
    print(char, end=' ')
```

Output :

```
P y t h o n
```

② Using range and len (for i in range(len(s)))

```
s = "Python"
for i in range(len(s)):
    print(i, s[i])
```

Output :

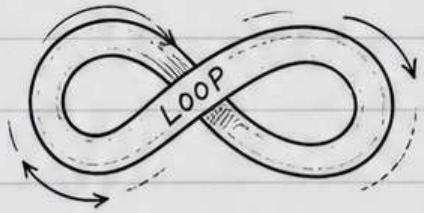
```
0 P
1 y
2 t
3 h
4 o
5 n
```

Pro Tip ☆

Use the first way for simplicity, the second way if you need the index !

Mini-Exercise

Write a loop to count vowels in a string.

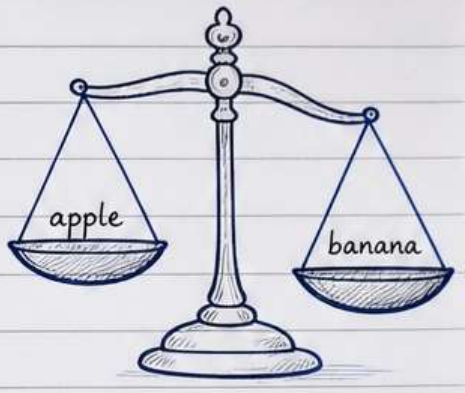


S.P Group of Institute

Date : 20/05/2025
Page No. : 23

Topic : String Comparison & Sorting

* Comparing Strings



In programming, strings are compared based on the ASCII (or Unicode) values of their characters.

- Comparison starts from the first character.
- If characters are same, it compares the next character, and so on.
- Shorter string is considered smaller if all compared characters are same.

Example : 'apple' < 'banana' (because 'a' < 'b')

Comparison Operators for Strings

==	: Equal to
!=	: Not equal to
<	: Less than
>	: Greater than
<=	: Less than or equal to
>=	: Greater than or equal to

Quick Revision : ASCII Values

Character	ASCII Value	Character	ASCII Value
A	65	a	97
B	66	b	98
C	67	c	99
...	...	...	...

(ASCII is case-sensitive)

Code Examples :

1.
```
print('A' > 'a')   # Compare characters
Output : False
```

2.
```
print('Apple' < 'banana')   # Compare strings
Output : True
```

3.
```
print('cat' == 'cat')
Output : True
```

Memory Trick

Uppercase letters come before lowercase in the ASCII world!

☺

* Note : ASCII comparison works for most characters. For special characters or different languages, Unicode is used.

S.P Group of Institute

Date : 18/05/2025

Page No. : 01

Practice Questions (Set 1)

* Practice Questions – Level: Beginner *

1) Reverse a string without slicing.

Hint :
Use a loop to swap characters from start and end.

2) Count occurrences of a character.

Hint :
Traverse the string and increment count when match is found.

3) Check if a string is a palindrome.

Hint :
Compare characters from start and end moving towards center.

4) Remove all spaces from a string.

Hint :
Build a new string and add only non-space characters.

5) Toggle case of each character (Uppercase to lowercase and vice versa).

Hint :
Use ASCII values or built-in functions like isupper(), islower().

★ **Challenge :**

6) Find the first non-repeating character in a string.

Hint :
Use a dictionary to store frequency of each character and return the first with count = 1.

★ Keep Practicing, Keep Improving! ★

S.P Group of Institute

Date : 25/05/2024
Page No. : 01

Topic : Interview Questions (Part 1)

Top Interview Questions on Strings

① What is string immutability?

- Strings in Python are immutable, meaning their content cannot be changed after creation.
- Any operation that modifies a string actually creates a new string.
- Example: s = "hello" s[0] = 'H' ✗ (Not allowed)

Pro Tip

Interviewers love to ask about immutability!

② Difference between find() and index()?

- Both methods search for a substring in a string.
- find() returns -1 if the substring is not found.
- index() raises a ValueError if the substring is not found.
- Example:

```
s = "hello world"
s.find("x")  → -1
s.index("x") → ValueError
```

③ How do f-strings work?

- f-strings (formatted string literals) allow embedding expressions inside string using { }.
- They are concise and more readable.
- Example:

```
name = "Alice"  age = 25
f"My name is {name} and I am {age} years old."
```

④ What is the difference between '==' and 'is' for strings?

- '==" checks value equality.
- 'is' checks object identity (same memory location).
- For strings, '==" should be used for comparison.
- Example: "abc" == "abc" → True
  "abc" is "abc" → Implementation dependent

S.P Group of Institute

Date : 24/05/24
Page No. : 42

Topic : Real-world Examples

✦ Strings in the Real World ✦

① Email validation (checking for @ and .)

```
def is_valid_email(email):
    return '@' in email and '.' in email

email = "user@example.com"
print(is_valid_email(email))   # True
```

Why it matters:
Helps ensure that users enter proper email formats before storing or sending important messages.

② Password strength checker (checking for digits and case)

```
def is_strong_password(pw):
    has_digit = any(c.isdigit() for c in pw)
    has_upper = any(c.isupper() for c in pw)
    has_lower = any(c.islower() for c in pw)
    return has_digit and has_upper and has_lower

pwd = "Abc123xyz"
print(is_strong_password(pwd))   # True
```

Why it matters:
Stronger passwords protect user accounts from brute-force attacks and unauthorized access.

③ URL parsing (splitting by /)

```
def parse_url(url):
    parts = url.split('/')
    return parts

url = "https://www.example.com/blog/post"
print(parse_url(url))
# ['https:', '', 'www.example.com', 'blog', 'post"]
```

Why it matters:
Useful for breaking down URLs to analyze domains, paths, or resources in web applications.

⚠ Common Mistake (Security Alert!)

Not stripping whitespace from passwords can lead to login failures or security loopholes.

Example:

```
pw = input("Enter password: ")      # Wrong → checks spaces too
# Right →  pw = pw.strip()          # removes leading/trailing spaces
```

Always clean and validate user input!

Code is powerful, but good logic makes it meaningful. ☺

Topic : Advanced String Methods & Regex Intro

1. Advanced String Methods

- replace(old, new [, count])
  → Returns a copy of the string with all occurrences of substring old replaced by new.

  Example :

```
text = "hello world"
text.replace("world", "Python")   # 'hello Python'
```

- zfill(width)
  → Pads the string on the left with zeros until it reaches the given width.

  Example :

```
'42'.zfill(5)   # '00042'
```

- expandtabs(tabsize = 8)
  → Replaces tab characters (\t) with spaces. Default tab size is 8.

  Example :

```
'a\tb'.expandtabs()   # 'a       b'
```

Sneak Peek : Regular Expressions (re module)

- re.search(pattern, string)
  → Searches for the first match of pattern in string.

  Example :

```
import re
txt = "Python is fun!"
re.search(r"fun", txt)   # <re.Match object; span=(10, 13), match='fun'>
```

- re.findall(pattern, string)
  → Returns a list of all non-overlapping matches of pattern.

  Example :

```
re.findall(r"\d+", "Roll no. 101 and 202")   # ['101', '202']
```

Pro Tip

Regex is like a superpower for string manipulation!

S.P Group of Institute

Date : 20 / 05 / 24
Page No. : 47

Topic : Quick Revision Summary

Quick Revision - Unit 5: Strings

STRINGS

1. Basics

- Strings are sequences of characters.
- Enclosed in single (' ') or double (" ") quotes.
- Example : "Hello World"
- Immutable (cannot be changed)

2. Indexing

H	e	l	l	o
0	1	2	3	4
-5	-4	-3	-2	-1

- Access characters using index.
- Forward Indexing : 0 to n-1
- Backward Indexing : -n to -1
- Example : s = "Hello"
  s[0] = 'H', s[-1] = 'o'

3. Slicing

- Extract part of a string.
- Syntax : s[start : end : step]
- Default : start=0, end = len(s), step = 1
- Example : s = "Hello World"
  s[0:5] → 'Hello'
  s[6:] → 'World'
  s[::-1] → 'dlroW olleH"

4. Methods

- len(s) → length of string
- lower() / upper() → case conversion
- strip() → remove spaces
- replace (old, new) → replace substring
- split (sep) → split into list
- join (iterable) → join list into string
- find (sub) → first index of substring
- count (sub) → count occurrences

5. Formatting

- f-strings : f"Name: {name}"
- .format() : "Name : {}". format(name)
- % formatting : "Name : %s" % name
- Example :

```
age = 21
print(f"I am {age} years old.")
```

6. Operations

- Concatenation : s1 + s2
- Repetition : s * n
- Membership : 'a' in s
- Comparison : s_1 == s2, s_1 != s2

Strings are powerful in handling text and are used in almost every program!

KEY FORMULAS

Slicing

- s[start:end] → from start to end-1
- s[start:] → start to end
- s[:end] → 0 to end-1
- s[::step] → step can be +ve or -ve
- s[::-1] → reverse string

Common Methods

- len(s) → integer
- s.lower() → lowercase
- s.upper() → uppercase
- s.strip() → remove spaces
- s.replace('a','b') → replace a with b
- s.split(',') → list
- ''.join(list) → string

FINAL TIP

Practice coding these methods to master them!

S.P Group of Institute

Date : 24/05/2025
Page No. : 1

Final Practice Test

Unit 5 - Final Assessment

A. Multiple Choice Questions (1 mark each)

1. Which of the following is NOT a feature of Object-Oriented Programming?
   a) Encapsulation b) Inheritance c) Compilation d) Polymorphism

2. Which keyword is used to inherit a class in Java?
   a) extends b) inherit c) implements d) extends from

3. What is the default value of a boolean variable in Java?
   a) true b) false c) 0 d) null

4. Which of the following is used to handle exceptions in Java?
   a) try-catch b) catch-throw c) throw-catch d) try-finally

B. True / False (1 mark each)

5. Java does not support multiple inheritance. _____
6. The 'static' keyword is used for memory management. _____
7. Constructor overloading is not possible in Java. _____

C. Predict the Output (2 marks each)

8. What will be the output of the following code?

```
1. int x = 5;
2. if (x > 3) {
3.     System.out.print("Hello");
4. } else {
5.     System.out.print("World");
6. }
```

Output : _____

9. What will be the output of the following code?

```
1. for(int i = 1; i <= 3; i++) {
2.     System.out.print(i + " ");
3. }
```

Output : _____

10. What will be the output of the following code?

```
1. String s = "Java";
2. System.out.println(s.toLowerCase());
```

Output : _____

Good Luck!

Answer Key (for reference only)

A: 1-c, 2-a, 3-b, 4-a | B: 5-True, 6-True, 7-False | C: 8-Hello, 9. 1 2 3, 10.-Java