11/07/2025.

DSA.

Data structures & Algorithms.

Stage 1 : Absolute Basics (Ground zero)

A) Basic programming:

1) variables, data types.

*what is a vanable?

A variable is a named storage in mem. where you can hold data to use and modify during your program.

Mem. - big row of lockers; variable - one locker is 'x' & we can put a number in it.

*why do we need variables?

- To store i/p from the users.
- To store intermediate results in calc.
- To make code reusable & meaningful.

Instead of `printf ("The sum is 7");`

we use
```
int a=3, b=4;
printf ("The sum is %d", a+b);
```

*Variable Rules:

- Must start with a letter or underscore, not a digit.
- can contain letters, digits, underscore.
- case-sensitive (Age ≠ age).
- cannot use keywords as names (eg: int, for, while).

*What is a data type?

A data type defines what kind of data a variable can hold - and what operations you can do on it.

Think of data types like labels on the locker saying, only numbers fit here or only text fit here.

*Common data types:

Datatype	Example	Size	
int	10, -5, 0	4 bytes	
float	3.14, -0.0001	4 bytes	-less precision
double	3.1415926	8 bytes	-more precision.
char	'A' 'b' '$'	1 byte	
string	"Hello"	Varies	
bool.	true, false.	1 byte	

February / 2024

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29		

eg: in java: int age = 20;
double salary = 50000.50;
boolean isPlaced = true;

* Mem./Default values:
- In c/Java, uninitialized variables may hold garbage values.
- In python: Always initiazed when you assign.
- int → usually '0' if initialized explicity.

* Constants:
- If no need for the value to be changed:
eg: Java: final int MAX = 100;

* common mistakes to avoid:
- Forgetting to initialize → garbage or errors.
- Mixing types carelessly. eg: int a = 5;
float b = 2.0;
int c = a/b; //problematic.
- overflow (eg: storing a number too big for 'int')

2) Input/output:

* what is i/p & o/p:
Your prgm. needs to interact with the user:
- i/p → getting data from the user.
- o/p → showing results back to the user.

Java: scanner → i/p
system.out.print/println → o/p.

* o/p in java:
s.o.print → stays on same line.
s.o.println → moves to nextline after printing.

* Formatting o/p:
for formatted o/p, use 'printf':
java: int age = 21.
* s.o.printf ("you are %d ~~old~~ years old\n", age);

o/p: you are 21 years old.

December / 2023

S	M	T	W	T	F	S
31					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

Format specifiers	meaning
%d	integer
%f	floating point
%s	string.
%.2f	float with dec. point.

java:
```
double pi = 3.14159;
    s.o.prinf("pi = %.2f \n", pi);
      o/p: pi = 3.14.
```

*Input in java:

- Import java.util.scanner;
  ↳ class for import.
- scanner sc = new scanner(system.in);
  ↳ obj. for class scanner.

Method.	Reads.
nextInt()	int
nextDouble()	double
next()	single word
nextLine().	whole line

*common mistakes:

*. Mixing nextLine() after nextInt() or nextDouble().

java:
```
int age = sc.nextInt();
sc.nextLine(); //clears the buffer.
String name = sc.nextLine();
```

3) Loops (for, while, do-while).

*why do we need loops?

Instead of writing the same statement multiple times: we use a loop to repeat tasks without duplication.

```
eg: s.o.p("Hello")
        "      "
        .      .
```

Instead:
```
for (int i=0; i<3; i++){
        s.o.p ("Hello");
    }.
```

February / 2024

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29		

loops are the backbone of:

- Repeating tasks.
- Traversing arrays/strings.
- Simulations/calculations.

* Types of loops in java:

Type	when to use?
for	when you know exactly how many times to run.
while	when you don't know exactly how many times, but based on a condition.
do-while	same as while but guarantees atleast one execution

- for loop:
```
for (initialization; condition: update) {
    //body
};
```

- while loop:
```
while (condition) {
    //body.
};
```

- do-while loop:
```
do {
    // body
} while (condition).
```

→ loop runs atleast once, even if cond. is false.

4) conditional statements (if-else, switch)

* why do we need conditionals?

- We need to make decisions based on conditions.

Eg: score > = 40 → pass.
temp > 30 → Turn on AC etc.

* If, if-else, else if:

- Basic if:
```
if (cond) {
    // body
}
```

- If else:
```
if (cond) {
    // body
} else {
    // body
}
```

December / 2023

S	M	T	W	T	F	S
31					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

- else if ladder:

```
if (cond) {
    //body
}else if (cond) {
    //body
}
    .
    .
}else {
  //body.
}
```

* when you have multiple conditions

* Switch statement:

- Use when you're checking a single variable against multiple fixed values - cleaner than many if-else.

- syntax:

```
switch (variable) {
        case value1:
            //body.
            break;   x.
        case value 2:
            //body.
            break;
        default:    //optional but recommended.
            //body.
    }.
```

5] Functions and Recursions.

* why use functions?

A func. (or method in java) is a reusable block of code that performs a specific task.

Without funcs:

- You'd duplicate code everywhere.
- programs become messy and hard to maintain.

February / 2024

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29		

* Anatomy of a func in java.

General syntax:
```
returntype funcName (parameters) {
    //body.
    return ~~type~~ value; //if returnType ≠ void.
}.
```

eg: Java:
```
int add(int a, int b) {
    return a+b;
}.
```

* Types of functions:

- funcs with return, with parameters.
- funcs with return, without parameters.
- funcs without return, with parameters.
- funcs without return, without parameters.

Examples: 1. return ✓ parameters ✓

Eg:
```
int add(int a, int b) {
    return a+b;
}
```

2. return ✗ parameters ✓

Eg:
```
void greet (String name) {
    s.o.p ("Hello" + name);
}
```

3. return ✗ parameters ✗

Eg:
```
void sayHello() {
    s.o.p ("Hello");
}
```

4. return ✓ parameters ✗

Eg:
```
int getRandomNumber() {
    return 4;
}
```

* How to call a function?

- You can (invoke) it inside main():

java:
```
int sum = add (5,7);
s.o.p (sum);
greet ("Ani");
sayHello();
```

December / 2023

S	M	T	W	T	F	S
31					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

*What about static?

In java, main() is static.

So, if func. - static; we can invoke it from main() directly

if func. - non-static; we should create object for the *class & call through it.

*Recursion:

• A func. that calls itself.

eg: factorial.

```
public static int factorial (int n) {
    if (n==0 || n==1) {
        return 1;
    }
    return n * factorial (n-1);
}
```

*Recursion vs loop:

• Recursion: elegant but can be slow & memory heavy.

• loop: More efficient.

• use recursion when the problm. is naturally recursive.

(eg: tree traversal, factorial, fibonacci etc.).

6) Arrays (1D & 2D).

*why do we need Arrays?

• To store a collection of values, not just one.

• Instead of writing 10 seperate `int' variables like:

```
int a1,a2,a3 .... a10;
```

• you can use array:

```
int [ ] arr = new int [10];
```

*One dimensional array (1D).

• It's like a list - elements are stored in a single row.

Declaration

```
int [] arr = new int [5];
```

*or with Initialization.

```
int [] arr = {1,2,3,4,5};
```

Access :

• Index - starts at 0.
ends at length - 1.

February / 2024

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29		

```
eg: arr[0] = 10;
    s.o.p(arr[0]);
```

· Traversal:

```
for (int i = 0; i < arr.length; i++) {
    s.o.p (arr[i]);
}
```

· Enhanced loop

```
for(int val : arr) {
    s.o.p.(val);
}
```

* Two-dimensional array (2D).

It's like a table - rows & columns.

· Declaration: `int [ ][ ] mat = new int [3][3];`

· Initialization

```
int [ ][ ] mat = {{1,2,3},
                  {4,5,6},
                  {7,8,9}
                 };
```

· Access

```
mat [0][0] = 10;
s.o.p (mat [0][0]);
```

· Traversal:

```
for(int i = 0; i < mat.length; i++) {
    for (int j = 0; j < mat.length; j++) (
        s.o.p (mat [i][j] + "  ");
    }
    s.o.p();
}
```

* common mistakes:

· Accessing out of bound indices → throws ArrayIndexOutOfBoundsException.

7) Strings (basic operations)

Java: Strings → objects that represent sequence of characters.
→ they are immutable (cannot change once created).

* Creating strings:

```
String s1 = "Hello";
string s2 = new string ("World");
```

December / 2023

S	M	T	W	T	F	S
31					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

*Getting length

```
string s = "Ani";
s.o.p (s.length ());
```

*Substring

```
string s = "Hello World";
s.o.p (s.substring (0,5));   //Hello.
s.o.p (s.substring (6));     //world.
```

*concatenation.

```
string s1 = "Hello";
string s2 = "World";
s.o.p (s1 + "   " + s2);   //Hello World.
s.oop (s1.concat ("   " + s2));   //Hello World.
```

*comparison.

•using .equals():

```
string a = "abc";
string b = "abc";
s.o.p (a.equals(b));   //true.
```

•using == (Not recommended for content comparison).

```
s.o.p (a==b);
```

•Lexicographical comparison.

```
String a = "abc";
String b = "abc";
s.o.p (a.compareTo(b));   //-1
```