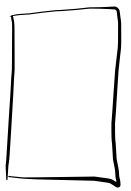
Basics of Programming

- I/O
- variables
- Decision
- Iterative
- operators

Functions

OOP



(modules)

mathematical functions

$$p(x, y) = x^2 + y^3 + 2x - 3y + 15$$

- parameters ✓
- returns a value/procedure ✓
- function Name ✓

$p(2)$ ✗

$$p(2,4) = (2)^2 + (4)^3 + 2(2) - 3(4) + 15$$

$$= 4 + 64 + 4 - 12 + 15$$

$$= 72 - 3$$

$$\Rightarrow 75$$

Functions

(int, String, char, float, boolean etc)

Type of value function is expected to return

Identifier

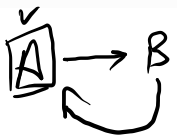
Input to function

```
public static  return-type  function-Name ( Parameter List )
{
    function body
}
```

Type 1: No return type

```
public static void sayHello () {
    System.out.println("Hello, greeting from me");
}
```

```java
import java.util.Scanner;
public class Main{
    public static void saysHello(){
        System.out.println("Hello from user !!");
    }

    public static int add(int v1,int v2){
        return v1+v2;
    }

    public static boolean isEven(int num){
        return (num%2)==0;
    }

    public static void main(String[] args) {
        saysHello();
        add(1646,365);
        //isEven(226);
    }
}
```



main
↳ special function
↳ calls itself

Hello from User!!

Use
↳ ① Debugging

Project
↳ fast → 1000 times + changes ⟹ =

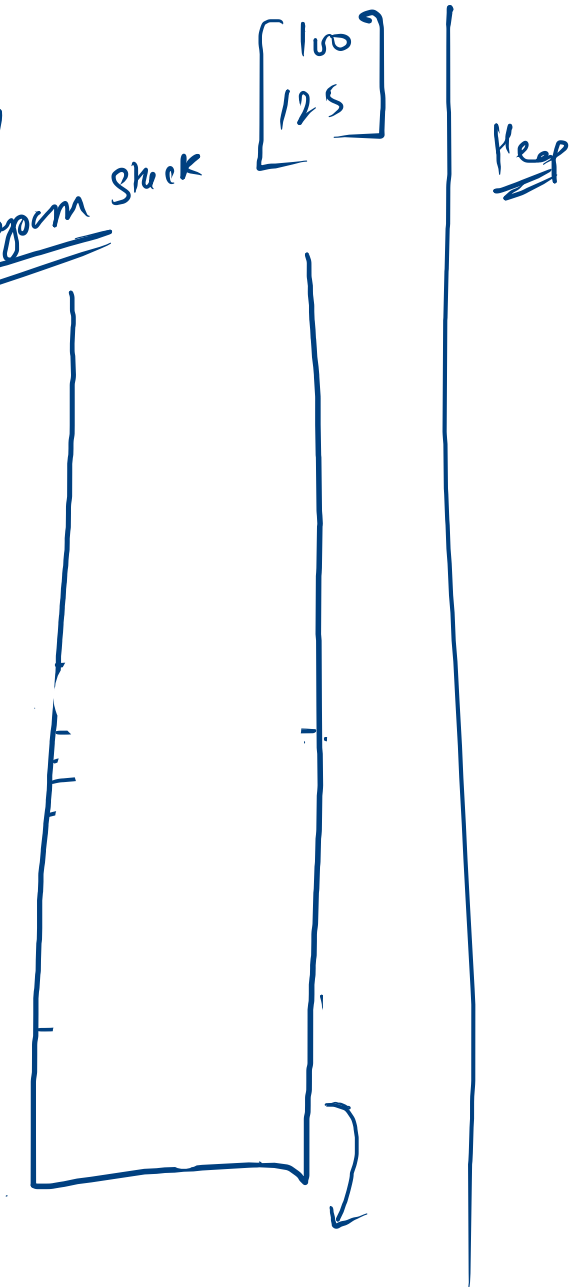
② Recursion

Function Overloading → using same name for function + different argument list (parameters)

Program Execution ⇒ 2 types of memory

Function call stack / system ⇒ Currently executing func looses its memory

```java
public static int area(int side){
    return side*side;
}
public static int area(int len,int bre){
    return len*bre;
}
public static void main(String[] args) {
    // saysHello();
    // System.out.println(add(1646,365));
    // isEven(226);

    int areaOfSq = area(10);
    int areaOfRect = area(5,25);

    System.out.println(areaOfSq);
    System.out.println(areaOfRect);
}
```



994543234

4

no. of occurances of digits in number

```
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int d = scn.nextInt();
    int f = getDigitFrequency(n, d);
    System.out.println(f);
}

public static int getDigitFrequency(int n, int d) {
    // write code here
}
```

Input

num

digit

Concept

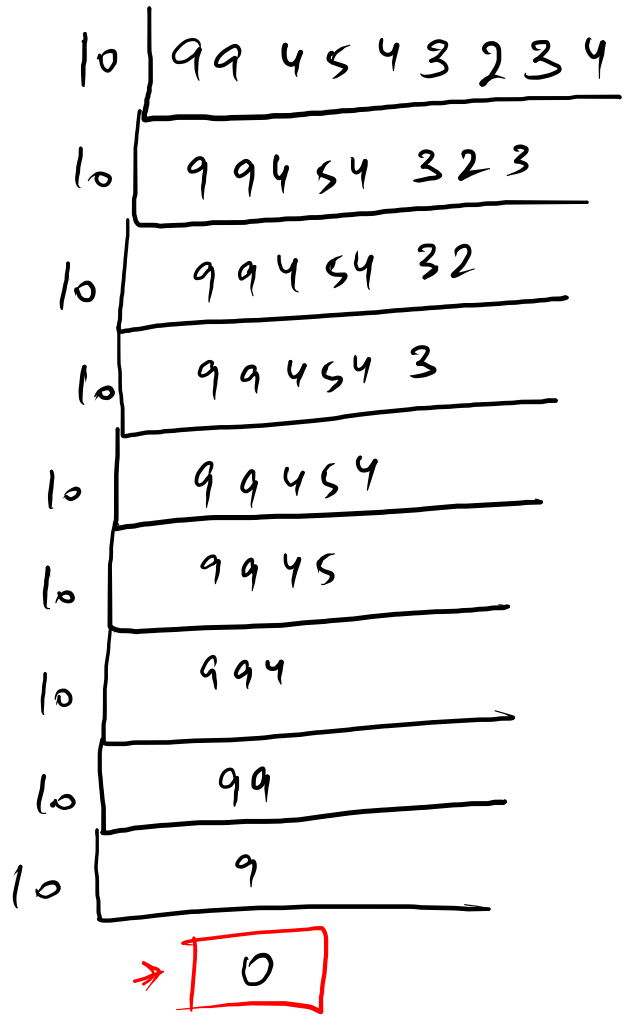
994543234

4

1-2 min

994543234 / 10 = 99454323 4

% 10 = 4



rem

4
3
2
3
4
5
4
9
9

digit = 4

Count = 0 → 1 → 2 → 3

```java
public static int getDigitFrequency(int n, int d) {
    int count = 0;

    while(n > 0){
        int rem = n % 10;
        if(rem == d){
            count += 1;
        }

        n = n / 10;
    }

    return count;
}
```

123121 1

n	rem	Count
~~123121~~	1	~~0~~
~~12312~~	2	1
~~1231~~	1	~~2~~
~~123~~	3	3
~~12~~	2	
~~1~~	1	
0		

Number Systems (Representation of no.)

- Binary (0, 1)
- Octal (0, 1, 2, 3, 4, 5, 6, 7)
- Decimal (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
- Hexa decimal (0 – 9, A – F)

Decimal → Base = 10 [0,1,2,3,4,5,6,7,8,9]

↳ unique no. of digits that can be used to represent a no. in it's respective system

Ex: 12, 12345, 1100, 1200

Counting [_ _] = 100

00	10	20
01	11	
02	12	
03		
04	13	
05	14	
06	15	
07		
08	16	
09	17	
	18	
	19	

99)

00000 → 99999

[10 × 10 × 10 × 10 × 10] ⇒ 10^5

① Count of unique no. that can be generated → 10^5

② [_ _ _ _ _]

↳ Largest no. = ? [99999]

↳ smallest no. → [00000]

Binary → Base = 2

↳ {0, 1}

Counting

0	1	0	1	0	0	1	0	0	0	1	1	0	0	1	1
1	1	1	1	0	1	1	0	0	1	1	1	0	1	0	1
			1	1	0	1	0	1	0	1	0	1	1	1	0
			1	1	1	1	0	1	1	1	1	1	1	1	1

[(2)(2)(2)(2)(2)] → 2^5

↑ ↑ ↑ ↑ ↑

① Count of unique no. → 2^5

② [_ _ _ _ _]

↳ largest → 11111

↳ smallest → 00000

Octal ⇒ Base ⇒ 8

↳ { 0, 1, 2, 3, 4, 5, 6, 7 }

—	—	—	—	—	—		70	100
0	1 0	2 0	3 0	~				
1	1 1	2 1						
2	1 2	2 2						
3								
4	1 3	2 3						
5	1 4	2 4						
6	1 5	2 5						
7	1 6	2 6			~		77	
—	1 7	2 7	3 7					

[8 8 8 8 8 — — — — —]

↳ Count = 8^5

↳ max = 77777

↳ min ⇒ 00000

Hexa decimal

Base → 16

0	10	F0
1	11	F1
2	12	
3	13	
4	14	
5	15	
6	16	
7	17	
8	18	
9	19	F9
A	1A	FA
B	1B	
C	1C	
D	1D	
E	1E	
F	1F	FF

0 1 2 3 4 5 6 7 8 9 A B C D E F

10 11 12 13 14 15

16 16 16 16 16

Count → (16^5)

max → (FFFFF)

min → (00000)

* Any base → Decimal (10)

PLACE VALUE

$(111011)_2 = (59)_{10}$

111011 ⇒ $(1.2^5)+(1.2^4)+(1.2^3)+(0.2^2)+(1.2^1)+(1.2^0)$

543210

⇒ 32 + 16 + 8 + 0 + 2 + 1

⇒ 59

Counting

0 = 0	2	1	0	4	1	0	0	8	1	0	0	0			
1 = 1	3	1	1	5	1	0	1	9	1	0	0	1			
				6	1	1	0	10	1	0	1	0			
				7	1	1	1	11	1	0	1	1			
								12	1	1	0	0			
								13	1	1	0	1			
								14	1	1	1	0			
								15	1	1	1	1			

$(1A)_{16} \Rightarrow (26)_{10}$

1 0

⇒ $(1\times16^1)+(A\times16^0)$

⇒ 16 + 10 ⇒ 26

$(1101)_2 \Rightarrow (13)_{10}$

1101 ⇒ $(1.2^3)+(1.2^2)+(0.2^1)+(1.2^0)$ ⇒ 8 + 4 + 0 = 1

3210

⇒ 13

$(2364)_8 \Rightarrow (\quad)_{10}$

3210 8

$(2.8^3)+(3.8^2)+(6.8^1)+(4.8^0)$

1024 + 192 + 48 + 4 ⇒ 1268

n = 111001

5 4 3 2 1 0

b = 2

1110 / 10 = 0

		rem	pos
10	111001	1	0
10	11100	0	1
10	1110	0	2
10	111	1	3
10	11	1	4
10	1	1	5
	0		6

Math.pow(rem, 10)

32
16
8
4
2
1

or ⇒ $(1 \cdot 2^0) + (0 \cdot 2^1) + (0 \cdot 2^2) + (1 \cdot 2^3) + (1 \cdot 2^4) + (1 \cdot 2^5)$

⇒ 57

1101 2

```java
public static int getValueIndecimal(int n, int b){
    int pos = 0;
    int rv = 0;

    while(n > 0){
        int rem = n % 10;

        rv = rv + (rem*(int)Math.pow(b,pos));

        pos += 1;
        n = n/10;
    }

    return rv;
}
```

n	rem	Pos	Pow
1101	1	0 →	1
		1 →	2
110	0		4
11	1	2	8
1	1	3	16
0			

$(1101)_2 \rightarrow (\ )_{10}$

$rv = (1 \cdot 2^0) + (0 \cdot 2^1) + (1 \cdot 2^2) + (1 \cdot 2^3) \rightarrow 13$

Decimal → Any base

$(57)_{10} \rightarrow (\ \ )_2$ $(111001)_2$

2 | 57		↑
2 | 28	1	
2 | 14	0	
2 | 7	0	
2 | 3	1	
2 | 1	1	
0	1	

$(125)_{10} \rightarrow (\ \ )_8$ $(175)_8$

8 | 125		↑
8 | 15	5	
8 | 1	7	
0	1	

1. $(1507)_{10} \rightarrow (\ \ )_2$ $(10\ 1111\ 000\ 11)_2$

2 | 1507		↑
2 | 753	1	
2 | 376	1	
2 | 188	0	
2 | 94	0	
2 | 47	0	
2 | 23	1	
2 | 11	1	
2 | 5	1	
2 | 2	1	
2 | 1	0	
0	1	

9 → ✗

2. $(780)_{10} \rightarrow (\ \ )_8 (1414)_8$

8 | 780		↑
8 | 97	4	
8 | 12	1	
8 | 1	4	
0	1	

3. $(100)_{10} \rightarrow (\ \ )_4 (1210)_4$

4 | 100		↑
4 | 25	0	
4 | 6	1	
4 | 1	2	
0	1	

$(780)_{10} \rightarrow (\quad)_8$

num

b

```
rem ← n % b
n = n / b
```

8	780
8	97
8	12
8	1
	0

rem	bow
4	$\times 10^0$
1	$\times 10^1$
4	$\times 10^2$
1	$\times 10^3$

4
10
400
1000

1414

$(780)_{10} \rightarrow (\quad)_8$ = n = 780, b = 8

		rem	pow
8	780	4	1 (10^0)
8	97	1	10 (10^1)
8	12	4	100 (10^2)
8	1	1	1000 (10^3)
	0		10000 (10^4)

```
int rv = 0;
int pow = 1;

while(n > 0){
    int rem = n%b;
    rv = rv + (rem*pow);
    pow = pow*10;
    n = n / b;
}

return rv;
```

rv = 0 + (4x1) + (1x10) + (4x100) + (1x1000) = => 1414

$(780)_{10} \rightarrow (\ )_8$

n b

		rem	pow	
8	780			
8	97	4	1	$[10^0]$
	12	1	10	$[10^1]$
			100	$[10^2]$

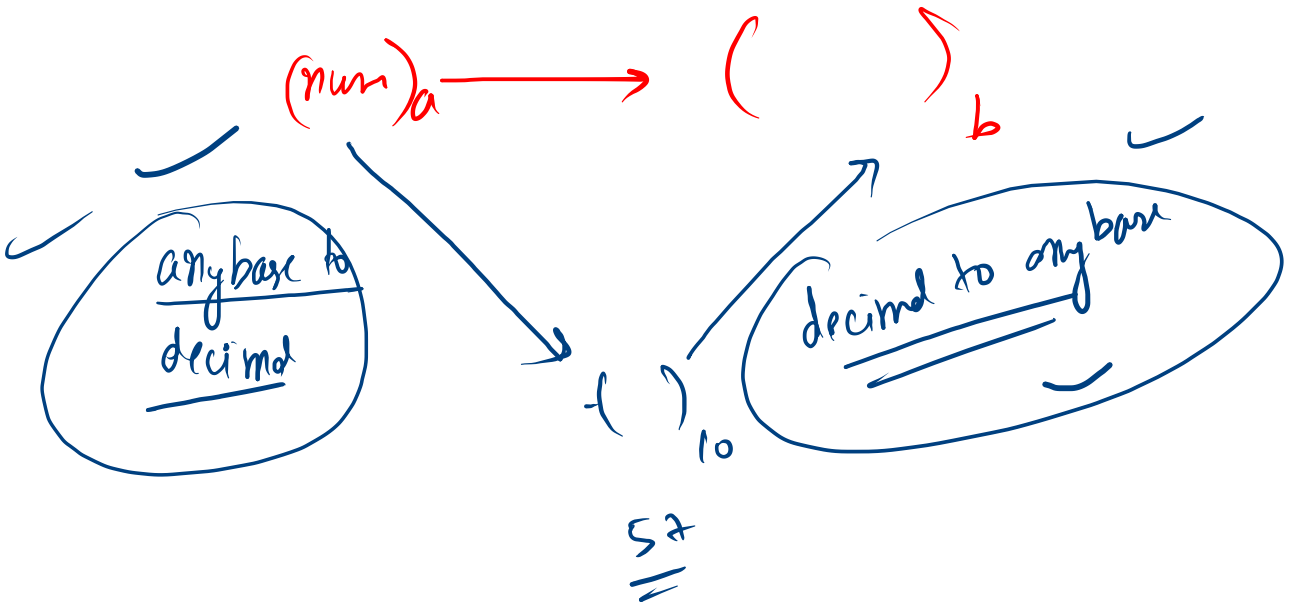
```
int rv = 0;
int pow = 1;

while(n > 0){
    int rem = n%b;
    rv = rv + (rem*pow);
    pow = pow*10;
    n = n / b;
}

return rv;
```

rv = 0 + 4 + 10

num = 111001

(num) b_1 = 2

(dos) b_2 = 3



DS → Basics

```
  1 1 1
  9 9 9
+ 8 8 7
-------
```

1 18/10 18/10 16/10 ⇒ carry

18/10 18/10 16/10 = 3 digit

1	8	8	6

Try it yourself

H.W. =

777
1

b = 8

```
    1 1 1
      7 7 7
    +   1
  ---------
  | 1 0 0 0 |
```

```
      0 1
      5 1 3
    +   2 1 6
  ---------
  | 7 3   1 |
```

Carry = Sum / base

digit = Sum % base

(0, 1, 2, 3, 4)

base = 5

```
    1 1 1
      3 4 3
    +   2 2 2
  ---------
  | 1 1 2 0 |
```

```
    1 1 1
      1 3 4 4
    +   4 3 2
  ---------
  | 2 3 3 1 |
```