CONTROL STATEMENT

a) Conditional Statements

i) if-else Statement

```
if (condition){
}
else{
}
```

ii) else if statements

```
if (condition1){
}
else if (condition2){
}
else{
}
```

iii) Ternary operator

variable = condition ? statement1 : statement2 ;
(True) (False)

Eg: int boolean larger = (5>3) ? 5 : 3 ; ⇒ 5

iv) Switch statement

```
switch (variable){
case 1:
    break;
case 2:
    break;
case 3:
    break;
default :
}
```

1/8/24 LOOPS

i) Whil loop

```
while (condition){
    // do something
}
```

ii) for loop

```
for (initialisation ; condition ; updation){
    // do something
}
```

* last digit = num % 10 => remainder
  remove last digit = num / 10 => (divide) quotient
* To get the reverse number

reverse = (reverse * 10) + last digit reverse = 0

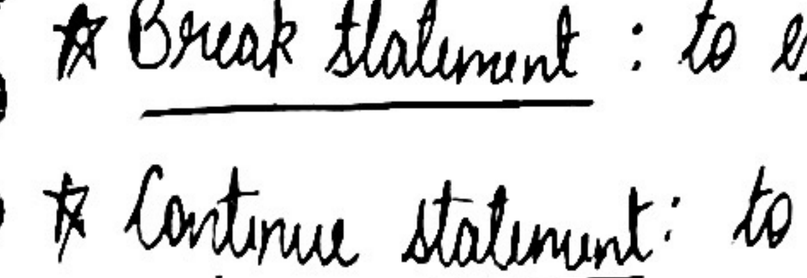
iii) do {

```
// do something
} while (condition);
```

☆ Break statement : to exit the loop

☆ Continue statement: to skip an iteration

* Prime number



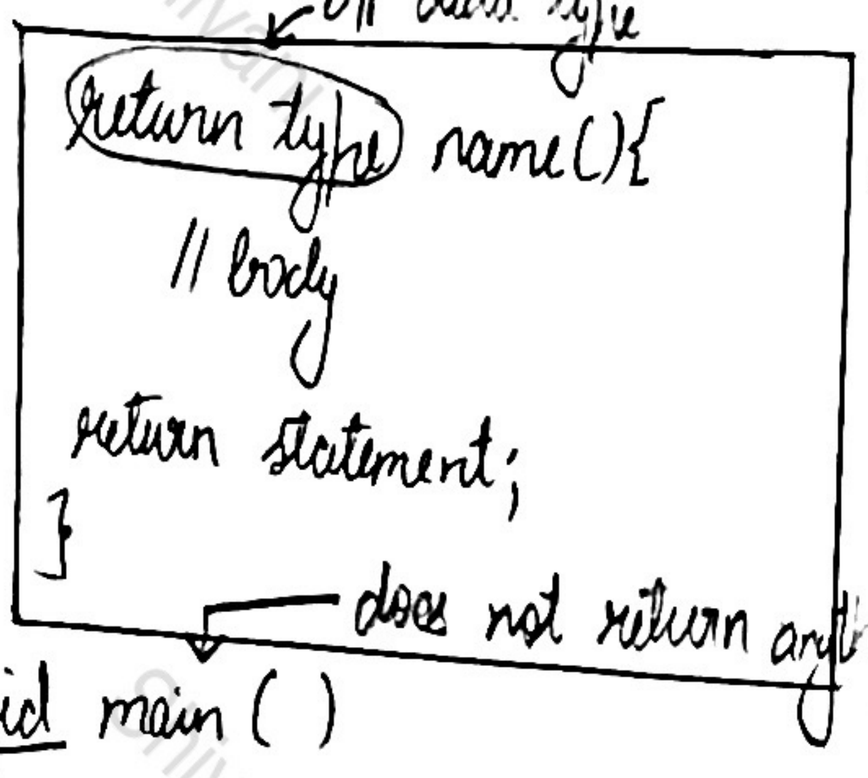

```
for (i=2, i<= math.sqrt(n); i++){
    if (n % i == 0){
        isPrime = false;
    }
}
```

* math.sqrt(n) ; $n = \sqrt{n} \times \sqrt{n}$

11/8/22

FUNCTIONS / METHODS

Syntax



func^n call => name();

<= func^n definition

* void main()
  ↳ returns nothing
* If return type is void then no need to write return in func^n def^n.
* func^n written in class is called method

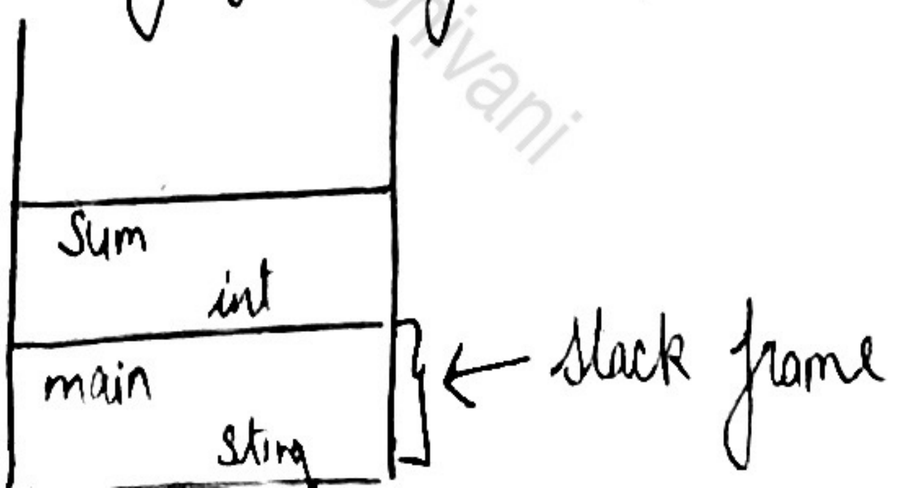
Syntax with parameters

```
return Type name(type param1, type param2){
    // body
return statement;
}
```

* parameters => formal parameters => written in func^n def^n
  arguments => actual parameters => written during func^n call

Memory

* call stack => tracks the calls
  each stack frame consist of func^n info (ip, variable)



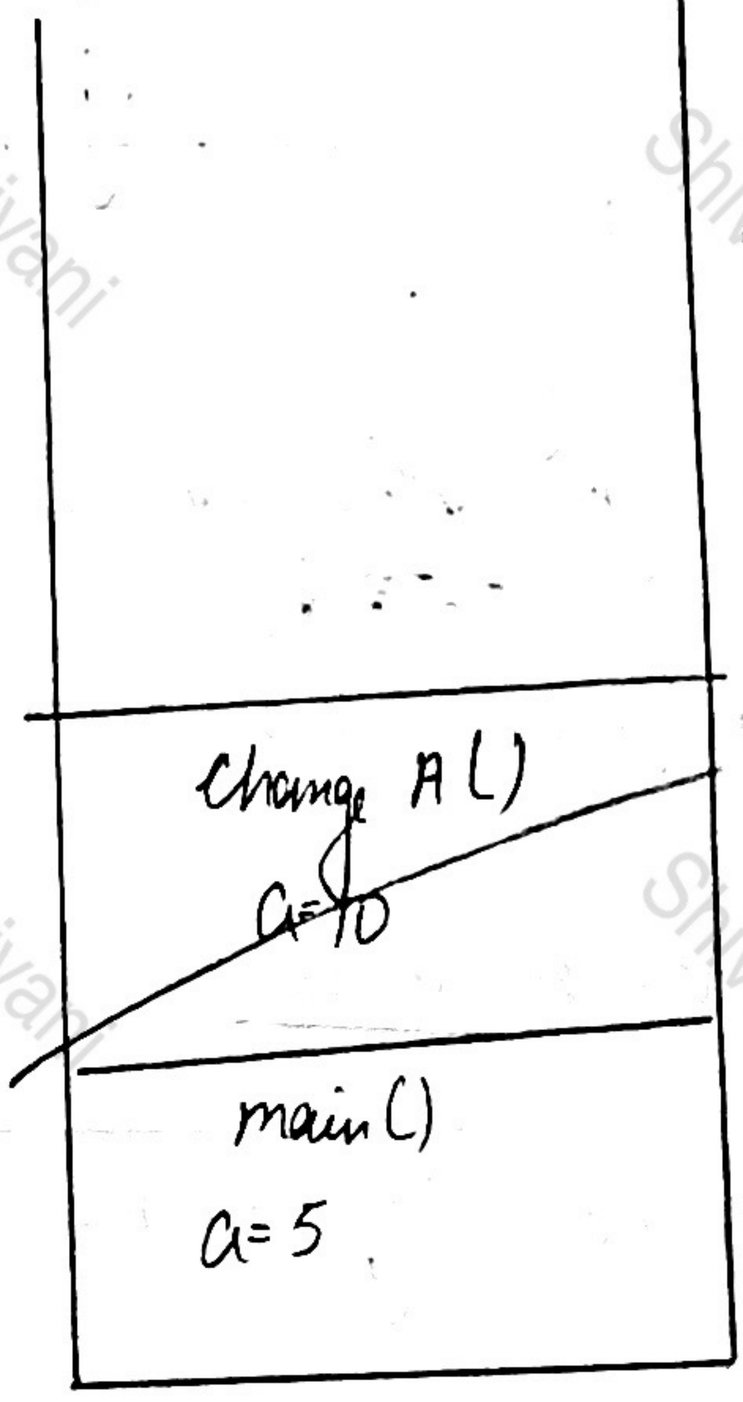
* When func" is called it occupies memory & when it encounters return statement, the memory is freed (released)

Call by Value

- Java always use call by value

Eg: change A (int a){
    a=10;
}

psvm (){
int a=5
change A(a);
sysO(a);
}



- copy of variable is passed.

* call by reference: passes original value

Methods

User defined

Inbuilt methods

math — pow
     — sqrt
     — max

sc.nextInt()

Func^ Overloading

Multiple functions with the same name but different parameters

Eg: Multiply (int a, int b) } using
Multiply (float a, float b) } data types

sum (int a, int b) } using
sum (int a, int b, int c) } parameters

→ type of parameters
→ numbers of paramaters

* func" overloading does not differentiate on basis of return type