1.

An **Armstrong** number is a positive m-digit number that is equal to the sum of the mth powers of their digits. It is also known as **pluperfect**, or **Plus Perfect**, or **Narcissistic** number. It is an OEIS sequence **A005188**. Let’s understand it through an example.

### Armstrong Number Example

**1:** 11 = **1**

**2:** 21 = **2**

**3:** 31 = **3**

**153:** 13 + 53 + 33 = 1 + 125+ 27 = **153**

**125:** 13 + 23 + 53 = 1 + 8 + 125 = **134 (Not an Armstrong Number)**

**1634:** 14 + 64 + 34 + 44 = 1 + 1296 + 81 + 256 = **1643**

Similarly, we can check other number also.

The first few Armstrong numbers between 0 to 999 are **1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407.** Some other Armstrong numbers are 1634, 8208, 9474, 54748, 92727, 93084, 548834, 1741725, 4210818, 9800817, 9926315, 24678050, 24678051, 88593477, 146511208, 472335975, 534494836, 912985153, 4679307774, 32164049650, 32164049651.

Code:

1. **import** java.util.Scanner;
2. **import** java.lang.Math;
3. **public** **class** ArmstsrongNumberExample
4. {
5. //function to check if the number is Armstrong or not
6. **static** **boolean** isArmstrong(**int** n)
7. {
8. **int** temp, digits=0, last=0, sum=0;
9. //assigning n into a temp variable
10. temp=n;
11. //loop execute until the condition becomes false
12. **while**(temp>0)
13. {
14. temp = temp/10;
15. digits++;
16. }
17. temp = n;
18. **while**(temp>0)
19. {
20. //determines the last digit from the number
21. last = temp % 10;
22. //calculates the power of a number up to digit times and add the resultant to the sum variable
23. sum += (Math.pow(last, digits));
24. //removes the last digit
25. temp = temp/10;
26. }
27. //compares the sum with n
28. **if**(n==sum)
29. //returns if sum and n are equal
30. **return** **true**;
31. //returns false if sum and n are not equal
32. **else** **return** **false**;
33. }
34. //driver code
35. **public** **static** **void** main(String args[])
36. {
37. **int** num;
38. Scanner sc= **new** Scanner(System.in);
39. System.out.print("Enter the limit: ");
40. //reads the limit from the user
41. num=sc.nextInt();
42. System.out.println("Armstrong Number up to "+ num + " are: ");
43. **for**(**int** i=0; i<=num; i++)
44. //function calling
45. **if**(isArmstrong(i))
46. //prints the armstrong numbers
47. System.out.print(i+ ", ");
48. }
49. }

Output:

Enter the limit: 999

Armstrong Number up to 999 are:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407

2.

# Palindrome Program in Java

Palindrome number in java: A **palindrome number** is *a number that is same after reverse*. For example 545, 151, 34543, 343, 171, 48984 are the palindrome numbers. It can also be a string like LOL, MADAM etc.

## Palindrome number algorithm

* Get the number to check for palindrome
* Hold the number in temporary variable
* Reverse the number
* Compare the temporary number with reversed number
* If both numbers are same, print "palindrome number"
* Else print "not palindrome number"

Let's see the palindrome program in java. In this java program, we will get a number variable and check whether number is palindrome or not.

1. **class** PalindromeExample{
2. **public** **static** **void** main(String args[]){
3. **int** r,sum=0,temp;
4. **int** n=454;//It is the number variable to be checked for palindrome
6. temp=n;
7. **while**(n>0){
8. r=n%10; //getting remainder
9. sum=(sum\*10)+r;
10. n=n/10;
11. }
12. **if**(temp==sum)
13. System.out.println("palindrome number ");
14. **else**
15. System.out.println("not palindrome");
16. }
17. }

Output:

Palindrome number

3.

# Java Stack

The **stack** is a linear data structure that is used to store the collection of objects. It is based on **Last-In-First-Out** (LIFO). [Java collection](https://www.javatpoint.com/collections-in-java) framework provides many interfaces and classes to store the collection of objects. One of them is the **Stack class** that provides different operations such as push, pop, search, etc.

In this section, we will discuss the **Java Stack class**, its **methods,** and **implement** the stack data structure in a [Java program](https://www.javatpoint.com/java-programs). But before moving to the Java Stack class have a quick view of how the stack works.

The stack data structure has the two most important operations that are **push** and **pop**. The push operation inserts an element into the stack and pop operation removes an element from the top of the stack

## Stack Class Constructor

The Stack class contains only the **default constructor** that creates an empty stack.

1. **public** Stack()

## Creating a Stack

If we want to create a stack, first, import the java.util package and create an object of the Stack class.

1. Stack stk = **new** Stack();

## Methods of the Stack Class

We can perform push, pop, peek and search operation on the stack. The Java Stack class provides mainly five methods to perform these operations. Along with this, it also provides all the methods of the [Java Vector class](https://www.javatpoint.com/java-vector).

| **Method** | **Modifier and Type** | **Method Description** |
| --- | --- | --- |
| [empty()](https://www.javatpoint.com/java-stack#empty) | boolean | The method checks the stack is empty or not. |
| [push(E item)](https://www.javatpoint.com/java-stack#push) | E | The method pushes (insert) an element onto the top of the stack. |
| [pop()](https://www.javatpoint.com/java-stack#pop) | E | The method removes an element from the top of the stack and returns the same element as the value of that function. |
| [peek()](https://www.javatpoint.com/java-stack#peek) | E | The method looks at the top element of the stack without removing it. |
| [search(Object o)](https://www.javatpoint.com/java-stack#search) | int | The method searches the specified object and returns the position of the object. |

### a) Stack Class push() Method

The method inserts an item onto the top of the stack. It works the same as the method [addElement(item) method](https://www.javatpoint.com/java-vector-addelement-method) of the Vector class. It passes a parameter **item** to be pushed into the stack.

### b) Stack Class pop() Method

The method removes an object at the top of the stack and returns the same object. It throws **EmptyStackException** if the stack is empty.

Code:

1. **import** java.util.\*;
2. **public** **class** StackPushPopExample
3. {
4. **public** **static** **void** main(String args[])
5. {
6. //creating an object of Stack class
7. Stack <Integer> stk = **new** Stack<>();
8. System.out.println("stack: " + stk);
9. //pushing elements into the stack
10. pushelmnt(stk, 20);
11. pushelmnt(stk, 13);
12. pushelmnt(stk, 89);
13. pushelmnt(stk, 90);
14. pushelmnt(stk, 11);
15. pushelmnt(stk, 45);
16. pushelmnt(stk, 18);
17. //popping elements from the stack
18. popelmnt(stk);
19. popelmnt(stk);
20. //throws exception if the stack is empty
21. **try**
22. {
23. popelmnt(stk);
24. }
25. **catch** (EmptyStackException e)
26. {
27. System.out.println("empty stack");
28. }
29. }
30. //performing push operation
31. **static** **void** pushelmnt(Stack stk, **int** x)
32. {
33. //invoking push() method
34. stk.push(**new** Integer(x));
35. System.out.println("push -> " + x);
36. //prints modified stack
37. System.out.println("stack: " + stk);
38. }
39. //performing pop operation
40. **static** **void** popelmnt(Stack stk)
41. {
42. System.out.print("pop -> ");
43. //invoking pop() method
44. Integer x = (Integer) stk.pop();
45. System.out.println(x);
46. //prints modified stack
47. System.out.println("stack: " + stk);
48. }
49. }

**Output:**

stack: []

push -> 20

stack: [20]

push -> 13

stack: [20, 13]

push -> 89

stack: [20, 13, 89]

push -> 90

stack: [20, 13, 89, 90]

push -> 11

stack: [20, 13, 89, 90, 11]

push -> 45

stack: [20, 13, 89, 90, 11, 45]

push -> 18

stack: [20, 13, 89, 90, 11, 45, 18]

pop -> 18

stack: [20, 13, 89, 90, 11, 45]

pop -> 45

stack: [20, 13, 89, 90, 11]

pop -> 11

stack: [20, 13, 89, 90]

### 

### c) Stack Class peek() Method

It looks at the element that is at the top in the stack. It also throws **EmptyStackException** if the stack is empty.

Code:

1. **import** java.util.Stack;
2. **public** **class** StackPeekMethodExample
3. {
4. **public** **static** **void** main(String[] args)
5. {
6. Stack<String> stk= **new** Stack<>();
7. // pushing elements into Stack
8. stk.push("Apple");
9. stk.push("Grapes");
10. stk.push("Mango");
11. stk.push("Orange");
12. System.out.println("Stack: " + stk);
13. // Access element from the top of the stack
14. String fruits = stk.peek();
15. //prints stack
16. System.out.println("Element at top: " + fruits);
17. }
18. }

**Output:**

Stack: [Apple, Grapes, Mango, Orange]

Element at the top of the stack: Orange

### d) Stack Class search() Method

The method searches the object in the stack from the top. It parses a parameter that we want to search for. It returns the 1-based location of the object in the stack. Thes topmost object of the stack is considered at distance 1.

Suppose, o is an object in the stack that we want to search for. The method returns the distance from the top of the stack of the occurrence nearest the top of the stack. It uses **equals()** method to search an object in the stack.

**StackSearchMethodExample.java**

1. **import** java.util.Stack;
2. **public** **class** StackSearchMethodExample
3. {
4. **public** **static** **void** main(String[] args)
5. {
6. Stack<String> stk= **new** Stack<>();
7. //pushing elements into Stack
8. stk.push("Mac Book");
9. stk.push("HP");
10. stk.push("DELL");
11. stk.push("Asus");
12. System.out.println("Stack: " + stk);
13. // Search an element
14. **int** location = stk.search("HP");
15. System.out.println("Location of Dell: " + location);
16. }
17. }

### 

### 

### d) Size of the Stack

We can also find the size of the stack using the [size() method of the Vector class](https://www.javatpoint.com/java-vector-size-method). It returns the total number of elements (size of the stack) in the stack.

**StackSizeExample.java**

1. **import** java.util.Stack;
2. **public** **class** StackSizeExample
3. {
4. **public** **static** **void** main (String[] args)
5. {
6. Stack stk = **new** Stack();
7. stk.push(22);
8. stk.push(33);
9. stk.push(44);
10. stk.push(55);
11. stk.push(66);
12. // Checks the Stack is empty or not
13. **boolean** rslt=stk.empty();
14. System.out.println("Is the stack empty or not? " +rslt);
15. // Find the size of the Stack
16. **int** x=stk.size();
17. System.out.println("The stack size is: "+x);
18. }
19. }

**Output:**

Is the stack empty or not? false

The stack size is: 5

# **Queue Interface In Java**

The Queue interface is present in [java.util](https://www.geeksforgeeks.org/java-util-package-java/) package and extends the [Collection interface](https://www.geeksforgeeks.org/collections-in-java-2/) is used to hold the elements about to be processed in FIFO(First In First Out) order. It is an ordered list of objects with its use limited to inserting elements at the end of the list and deleting elements from the start of the list, (i.e.), it follows the **FIFO** or the First-In-First-Out principle.

**Example:** Queue

// Java program to demonstrate a Queue

import java.util.LinkedList;

import java.util.Queue;

public class QueueExample {

public static void main(String[] args)

{

Queue<Integer> q

= new LinkedList<>();

// Adds elements {0, 1, 2, 3, 4} to

// the queue

for (int i = 0; i < 5; i++)

q.add(i);

// Display contents of the queue.

System.out.println("Elements of queue "

+ q);

// To remove the head of queue.

int removedele = q.remove();

System.out.println("removed element-"

+ removedele);

System.out.println(q);

// To view the head of queue

int head = q.peek();

System.out.println("head of queue-"

+ head);

// Rest all methods of collection

// interface like size and contains

// can be used with this

// implementation.

int size = q.size();

System.out.println("Size of queue-"

+ size);

}

}

**Output:**

Elements of queue [0, 1, 2, 3, 4]

removed element-0

[1, 2, 3, 4]

head of queue-1

Size of queue-4

### **Operations on Queue Interface**

Let’s see how to perform a few frequently used operations on the queue using the [Priority Queue class](https://www.geeksforgeeks.org/priority-queue-class-in-java-2/).

**1. Adding Elements:** In order to add an element in a queue, we can use the [add() method](https://www.geeksforgeeks.org/queue-add-method-in-java/). The insertion order is not retained in the PriorityQueue. The elements are stored based on the priority order which is ascending by default.

code:

// Java program to add elements

// to a Queue

import java.util.\*;

public class GFG {

public static void main(String args[])

{

Queue<String> pq = new PriorityQueue<>();

pq.add("Geeks");

pq.add("For");

pq.add("Geeks");

System.out.println(pq);

}

}

**Output:**

[For, Geeks, Geeks]

**2. Removing Elements:** In order to remove an element from a queue, we can use the [remove() method.](https://www.geeksforgeeks.org/queue-remove-method-in-java/) If there are multiple such objects, then the first occurrence of the object is removed. Apart from that, poll() method is also used to remove the head and return it.

Code:

// Java program to remove elements

// from a Queue

import java.util.\*;

public class GFG {

public static void main(String args[])

{

Queue<String> pq = new PriorityQueue<>();

pq.add("Geeks");

pq.add("For");

pq.add("Geeks");

System.out.println("Initial Queue " + pq);

pq.remove("Geeks");

System.out.println("After Remove " + pq);

System.out.println("Poll Method " + pq.poll());

System.out.println("Final Queue " + pq);

}

}

**Output:**

Initial Queue [For, Geeks, Geeks]

After Remove [For, Geeks]

Poll Method For

Final Queue [Geeks]

**3. Iterating the Queue:** There are multiple ways to iterate through the Queue. The most famous way is converting the queue to the array and traversing using the for loop. However, the queue also has an inbuilt iterator which can be used to iterate through the queue.

Code:

// Java program to iterate elements

// to a Queue

import java.util.\*;

public class GFG {

public static void main(String args[])

{

Queue<String> pq = new PriorityQueue<>();

pq.add("Geeks");

pq.add("For");

pq.add("Geeks");

Iterator iterator = pq.iterator();

while (iterator.hasNext()) {

System.out.print(iterator.next() + " ");

}

}

}

**Output:**

For Geeks Geeks

5.

# Duck Number Java

Duck number is another special positive non-zero number that contains zero in it. The digit zero should not be presented at the starting of the number. Zero can be present at any of the positions except the beginning of the number.

Let's understand some examples of Duck numbers.

1. 3210 is a Duck number because it contains zero at the end of the number but not present at the beginning of it.
2. 08237 is not a Duck number because it contains zero at the beginning of it.
3. 7033 is a Duck number because it contains zero at the second position, not at the beginning.
4. 030405 is not a Duck number because it also contains zero in starting of the number.
5. 00153 is also not a Duck number because it contains leading zeros.

These are the following steps that we use to check whether the given number is a Duck number or not.

1. We first take a number.
2. We then find the last digit of the number.
3. If the last digit is zero, it is a Duck number.
4. Otherwise, we remove that digit from the number.
5. Perform steps 2, 3, and 4 until the number becomes zero.

**DuckNumber.java**

1. //import required classes and packages
2. **import** java.util.\*;
3. **import** java.io.\*;
4. **import** java.util.Scanner;
6. //create DuckNumberExample class to check whether the given number is a Duck number or not
7. **public** **class** DuckNumberExample {
9. // create checkNumber() method that returns true when it founds number Buzz
10. **public** **static** **boolean** checkNumber(**int** number) {
12. // use loop to repeat steps
13. **while**(number != 0) {
15. // check whether the last digit of the number is zero or not
16. **if**(number%10 == 0)
17. **return** **true**; //return true if the number is Duck
19. // divide the number by 10 to remove the last digit
20. number = number / 10;
21. }
23. **return** **false**; //return false if the number is not Duck
24. }
25. // main() method start
26. **public** **static** **void** main(String args[])
27. {
28. **int** n1, n2;
30. //create scanner class object to get input from user
31. Scanner sc=**new** Scanner(System.in);
33. //show custom message
34. System.out.println("Enter first number");
36. //store user entered value into variable n1
37. n1 = sc.nextInt();
39. //show custom message
40. System.out.println("Enter second number");
42. //store user entered value into variable n2
43. n2 = sc.nextInt();
45. **if** (checkNumber(n1))
46. System.out.println(n1 + " is a Duck number");
47. **else**
48. System.out.println(n1 + " is not a Duck number");
49. **if** (checkNumber(n2))
50. System.out.println(n2 + " is a Duck number");
51. **else**
52. System.out.println(n2 + " is not a Duck number");
53. }
54. }

**Output:**

Enter first number

00432

Enter second number

0340240

432 is not a Duck number

340240 is a Duck number

6.

# Magic Number in Java

In programming, a **magic number** is a numeric value that is used directly in the code. It is used for identification purposes. In this section, we will discuss **what is a magic number** and **how can we find a magic number through a Java program.**

## Magic Number in Programming

A **magic number** is a hard-coded numeric value (text value in some cases) in the code that may change at a later stage. It seems like arbitrary and has no context or meaning. It is hard to update. For example:

1. **const** num = 74;
2. //where 2 is hard-coded
3. **const** number = num / 2; //it should be properly defined

## Magic Number in Mathematics

In mathematics, if the sum of its digits **recursively** is calculated till a single digit. If the single digit is 1 then the number is called a **magic number**. It is quite similar to the **happy number**.

For example, **325** is a magic number because the sum of its digits (3+2+5) is 10, and again sum up the resultant (1+0), we get a single digit (1) as the result. Hence, the number 325 is a magic number.

Some other magic numbers are **1234, 226, 10, 1, 37, 46, 55, 73, etc**.

Note that if a number is a magic number then all the possible combinations of the number will also be the magic numbers.

For example, 532, 253, 325, 235, 352, 523 the sum of digits of all the numbers gives 10 and again sum up the resultant (1+0), we get a single-digit i.e. 1. Hence, we can say that the magic number and its combinations are also magic.

## Java Magic Number Program

**MagicNumberExample1.java**

1. **import** java.util.Scanner;
2. **public** **class** MagicNumberExample1
3. {
4. **public** **static** **void** main(String args[])
5. {
6. **int** n, remainder = 1, number, sum = 0;
7. //creating a constructor of the Scanner class
8. Scanner sc = **new** Scanner(System.in);
9. System.out.print("Enter a number you want to check: ");
10. //reading an integer form the user
11. n = sc.nextInt();
12. //assigning the entered number in the variable num
13. number = n;
14. //outer while loop
15. **while** (number > 9) //while(number > 0 || sum > 9)
16. {
17. //inner while loop
18. **while** (number > 0)
19. {
20. //determines the remainder
21. remainder = number % 10;
22. sum = sum + remainder;
23. //divides the number by 10 and removes the last digit of the number
24. number = number / 10;
25. }
26. number = sum;
27. sum = 0;
28. }
29. **if** (number == 1)
30. {
31. System.out.println("The given number is a magic number.");
32. }
33. **else**
34. {
35. System.out.println("The given number is not a magic number.");
36. }
37. }
38. }

**Output 1:**

Enter a number you want to check: 325

The given number is a magic number

**MagicNumberExample2.java**

1. **import** java.util.Scanner;
2. **public** **class** MagicNumberExample2
3. {
4. **public** **static** **void** main(String args[])
5. {
6. Scanner scanner = **new** Scanner(System.in);
7. System.out.print("Enter any number to check: ");
8. //reading an iteger from the user
9. **int** number = scanner.nextInt();
10. **if**(magicNumber(number))
11. System.out.println(number +" is a magic number.");
12. **else**
13. System.out.println(number +" is not a magic number.");
14. }
15. //user-defined method to check the number is magic or not
16. **public** **static** **boolean** magicNumber(**int** number)
17. {
18. **if**( ((number - 1) % 9) == 0)
19. **return** **true**;
20. **else**
21. **return** **false**;
22. }
23. }

**Output 1:**

**Enter any number to check: 73**

**73 is a magic number.**

**7.**

**Data driven calculator:**

**Code:**

**import java.util.Scanner;**

**public class CodesCracker**

**{**

**public static void main(String[] args)**

**{**

**double a, b, res;**

**int choice;**

**Scanner scan = new Scanner(System.in);**

**System.out.println("1. Addition");**

**System.out.println("2. Subtraction");**

**System.out.println("3. Multiplication");**

**System.out.println("4. Division");**

**System.out.println("5. nth Power");**

**System.out.println("6. nth Root");**

**System.out.print("Enter Your Choice (1-6): ");**

**choice = scan.nextInt();**

**if(choice>=1 && choice<=6)**

**{**

**if(choice==1){**

**System.out.print("\nEnter any Two Number: ");**

**a = scan.nextFloat();**

**b = scan.nextFloat();**

**res = a+b;**

**}**

**else if(choice==2)**

**{**

**System.out.print("\nEnter any Two Number: ");**

**a = scan.nextFloat();**

**b = scan.nextFloat();**

**res = a-b;**

**}**

**else if(choice==3)**

**{**

**System.out.print("\nEnter any Two Number: ");**

**a = scan.nextFloat();**

**b = scan.nextFloat();**

**res = a\*b;**

**}**

**else if(choice==4)**

**{**

**System.out.print("\nEnter any Two Number: ");**

**a = scan.nextFloat();**

**b = scan.nextFloat();**

**res = a/b;**

**}**

**else if(choice ==5)**

**{**

**System.out.print("\nEnter the Number whose power is to be raised: ");**

**a = scan.nextFloat();**

**System.out.print("\nEnter the n fot nth-power: ");**

**b = scan.nextInt();**

**res=Math.pow(a,b);**

**}**

**else{**

**System.out.println("\nEnter the number whose root is to be done");**

**a=scan.nextFloat();**

**System.out.println("\nEnter the n of nth-root");**

**b=scan.nextInt();**

**res=Math.pow(a,1/b);**

**}**

**System.out.println("\nResult = " +res);**

**}**

**else**

**System.out.println("\nInvalid Choice!");**

**}**

**}**

**Output:**

**1. Addition**

**2. Subtraction**

**3. Multiplication4. Division**

**5. nth Power**

**6. nth Root**

**Enter Your Choice (1-6): 6**

**Enter the number whose root is to be done**

**Enter the n of nth-root**

**5**

**2**

**Result = 2.23606797749979**

**10.**

**a.**

**import java.util.Scanner;**

**public class Edureka**

**{**

**public static void main(String[] args)**

**{**

**Scanner sc = new Scanner(System.in);**

**System.out.println("Enter the number of rows: ");**

**int rows = sc.nextInt();**

**for (int i= 0; i<= rows-1 ; i++)**

**{**

**for (int j=0; j<=i; j++)**

**{**

**System.out.print(" ");**

**}**

**for (int k=0; k<=rows-1-i; k++)**

**{**

**System.out.print("\*" + " ");**

**}**

**System.out.println();**

**}**

**sc.close();**

**}**

**}**

Output:

Enter the number of rows:

5

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

b)

import java.util.Scanner;

public class Pyramid{

public static void main(String args[]){

int i, j, my\_input;

System.out.println("Required packages have been imported");

Scanner my\_scanner = new Scanner(System.in);

System.out.println("A reader object has been defined ");

System.out.print("Enter the number of rows : ");

my\_input = my\_scanner.nextInt();

System.out.println("The pyramid star pattern : ");

for (i=0; i<my\_input; i++){

for (j=my\_input-i; j>1; j--){

System.out.print(" ");

}

for (j=0; j<=i; j++ ){

System.out.print("\* ");

}

System.out.println();

}

}

}

## Output

Required packages have been imported

A reader object has been defined

Enter the number of rows : 8

The pyramid star pattern :

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

\* \* \* \* \* \*

\* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

**c)**

import java.util.Scanner;

public class RightTriangleNumber1 {

private static Scanner sc;

public static void main(String[] args) {

sc = new Scanner(System.in);

System.out.print("Right Triangle Number Pattern Rows = ");

int rows = sc.nextInt();

System.out.println("Right Angled Triangle Number Pattern");

for (int i = 1 ; i <= rows; i++ )

{

for (int j = 1 ; j <= i; j++ )

{

System.out.printf("%d ", i);

}

System.out.println();

}

}

}

output:

Right Triangle Number Pattern Rows = 5

Right Angled Triangle Number Pattern

1

2 2

3 3 3

4 4 4 4

5 5 5 5 5

d)

import java.util.Scanner;

public class Exercise16 {

public static void main(String[] args)

{

int i,j,n;

System.out.print("Input number of rows : ");

Scanner in = new Scanner(System.in);

n = in.nextInt();

for(i=1;i<=n;i++)

{

for(j=1;j<=i;j++)

System.out.print(j);

System.out.println("");

}

}

}

output:

Input number of rows : 6

1

12

123

1234

12345

123456

e)

import java.util.Scanner;

public class Pattern14

{

public static void main(String[] args)

{

int i, j, rows;

Scanner sc = new Scanner(System.in);

System.out.print("Enter the number of rows you want to print: ");

rows = sc.nextInt();

for (i = rows; i >= 1; i--)

{

for (j = 1; j <= i; j++)

{

System.out.print(j+" ");

}

System.out.println();

}

}

}

Output:

Enter the number of rows you want to print: 5

1 2 3 4 5

1 2 3 4

1 2 3

1 2

1

f)

import java.util.Scanner;

public class Edureka

{

public static void main(String[] args) {

int i, j, k = 1,row;

Scanner sc = new Scanner(System.in);

System.out.print("Enter the number of rows you want to print: ");

row = sc.nextInt();

for (i = 1; i <= row; i++) {

for (j = 1; j< i + 1; j++) {

System.out.print(k++ + " ");

}

System.out.println();

}

}

}

Output:

Enter the number of rows you want to print:3

1

23

456

8.

Area and perimeter:

a.rectangle

import java.util.Scanner;

public class Rectangle {

public static void main(String[] args) {

float length, width, area, perimeter;

// Create scanner class object

Scanner in = new Scanner(System.in);

// Input length and width of rectangle

System.out.print("Enter length of rectangle: ");

length = in.nextFloat();

System.out.print("Enter width of rectangle: ");

width = in.nextFloat();

// Calculate perimeter of rectangle

perimeter = 2 \* (length + width);

// Calculate area of rectangle

area = length \* width;

// Print perimeter and area of rectangle

System.out.println("Perimeter of rectangle is " + perimeter + " units.");

System.out.println("Area of rectangle is " + area + " sq. units.");

}

}

output:

Enter length of rectangle: 12

Enter width of rectangle: 13

Perimeter of rectangle is 50.0 units.

Area of rectangle is 156.0 sq. units.

B.circle

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner io = new Scanner(System.in);

System.out.println("Input the radius of the circle: ");

double radius = io.nextDouble();

System.out.println("Perimeter is = " + (2 \* radius \* Math.PI));

System.out.println("Area is = " + (Math.PI \* radius \* radius));

}

}

Sample Output:

Perimeter is = 47.12388980384689

Area is = 176.71458676442586

C.square

import java.util.Scanner;

public class AreaAndPerimeterOfSquare {

public static void main(String[] args) {

try (Scanner scanner = new Scanner(System.in)) {

System.out.printf("1. Enter side of square : ");

double side = scanner.nextDouble();

//Area of square is side \* side

double area = side \* side;

//Print area up to two precision

System.out.printf("2. Area of square is : %4.2f",area);

//Area of square is side \* side

double Perimeter = 4 \* side;

//Print area up to two precision

System.out.printf("\n3. Perimeter of square is : %4.2f",Perimeter);

}

}

}

Output:

1. Enter side of square : 2

2. Area of square is : 4.003.

Perimeter of square is : 8.00

D.triangle

import java.util.Scanner;

public class AreaOfTriangle {

private static Scanner sc;

public static void main(String[] args) {

double a, b, c, Perimeter, s, Area;

sc = new Scanner(System.in);

System.out.println("\n Please Enter Three sides of triangle: ");

a = sc.nextDouble();

b = sc.nextDouble();

c = sc.nextDouble();

Perimeter = a + b + c;

s = (a + b + c)/2; // Perimeter/2

Area = Math.sqrt(s\*(s-a)\*(s-b)\*(s-c));

System.out.format("\n The Perimeter of Traiangle = %.2f\n", Perimeter);

System.out.format("\n The Semi Perimeter of Traiangle = %.2f\n",s);

System.out.format("\n The Area of triangle = %.2f\n",Area);

}

}

Output:

Please Enter Three sides of triangle: 3

4

5

The Perimeter of Traiangle = 12.00The Semi Perimeter of Traiangle = 6.00

The Area of triangle = 6.00