Diploma in

Computer Science

Loops and Functions

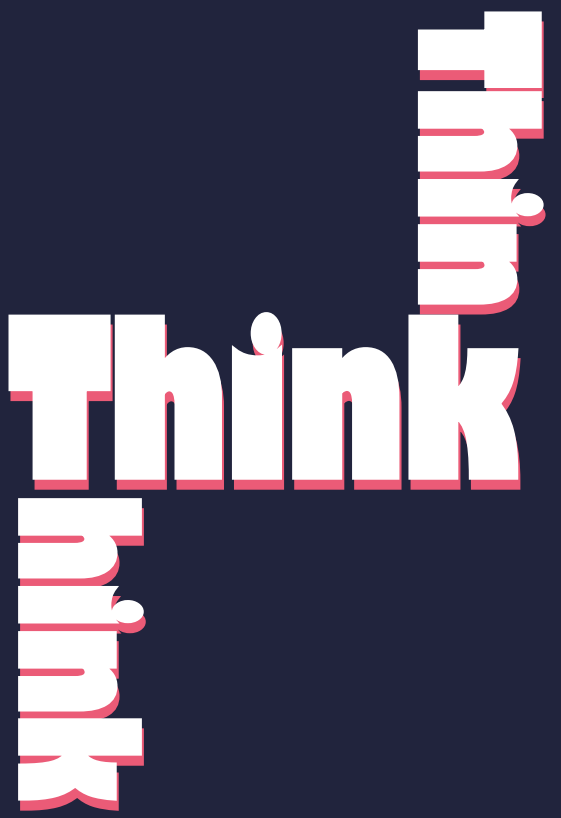
Objectives

- Recognise how computers achieve repetition
- Discuss the methods for carrying out repetitive tasks
- Explore common repetitive programs
- Appreciate the common mistakes

Repetition

Three types of repetition

- While and do-while loops
- For loops
- Recursion



While loop

Structure used to repetitively execute a specific block of code until a condition is met

Code is enclosed in curly brackets

Can only be run from within the while loop

Loops in everyday life

While loop

Controlled by a Boolean statement

Boolean statement is evaluated whenever the loop ‘goes over’ itself

While loop

Presented with condition

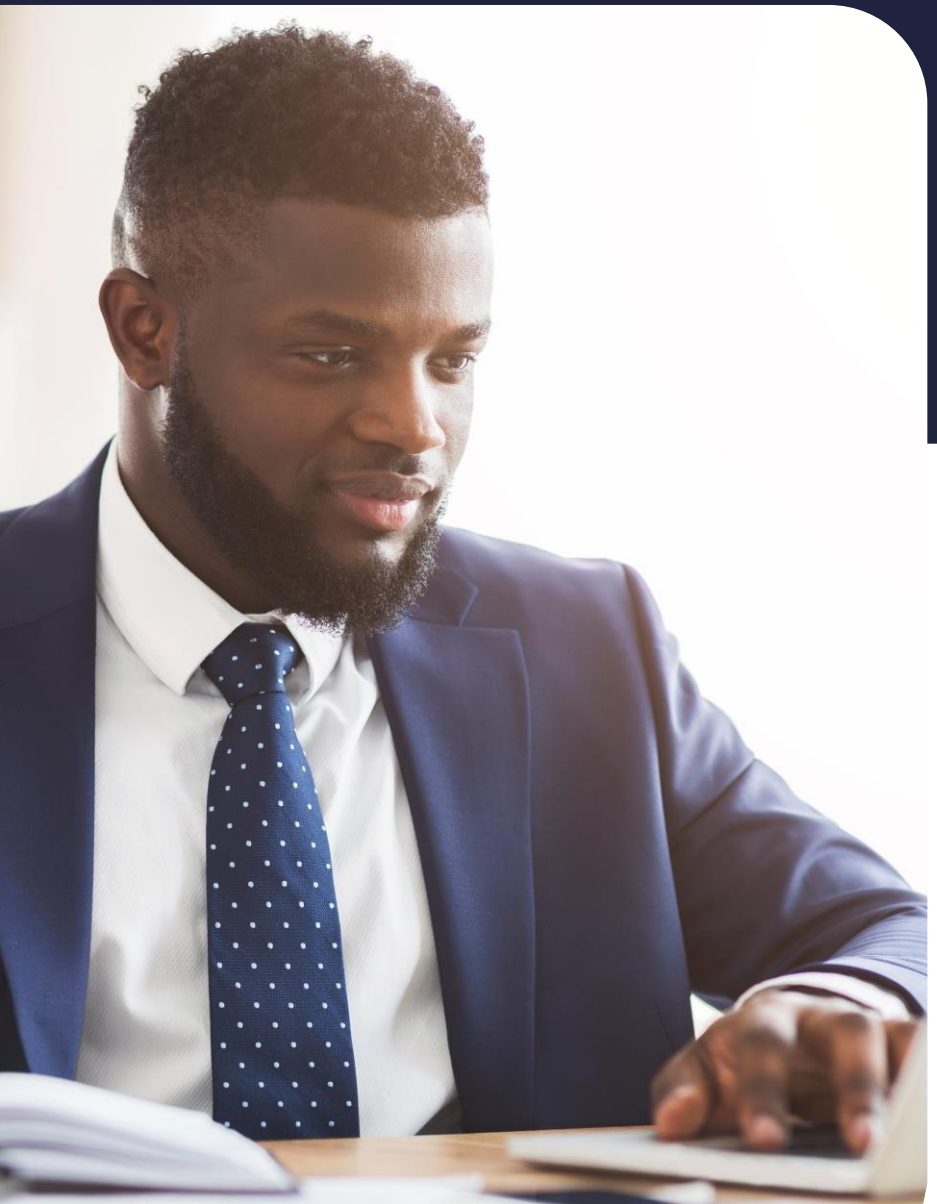
At ever iteration condition is evaluated

Proceeds if true

Condition is altered

Returns to condition to re-evaluate it

Loop is dropped when no longer true

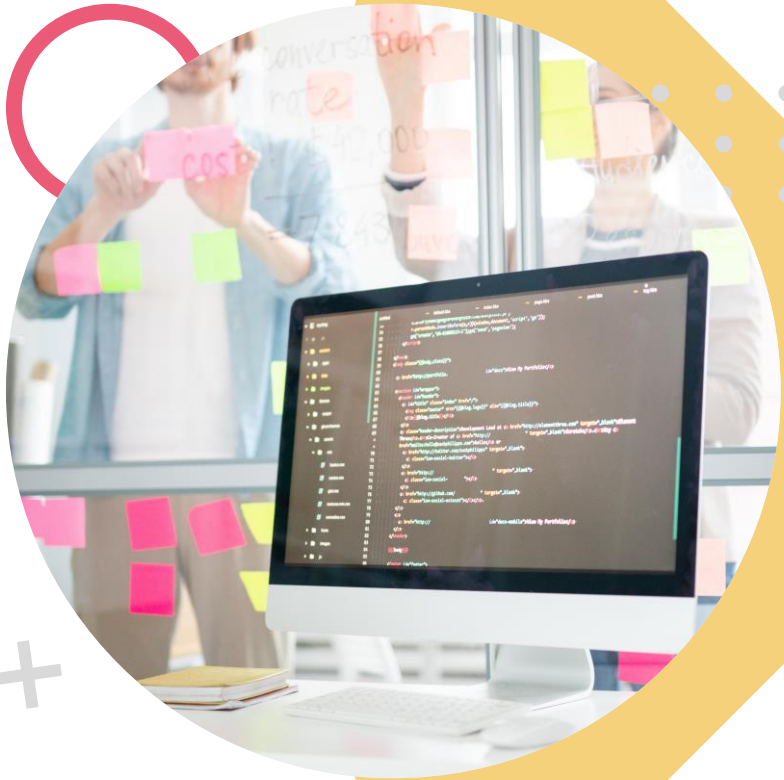


While loop – control variable

Variable used in the condition

To start must be of a certain desirable value

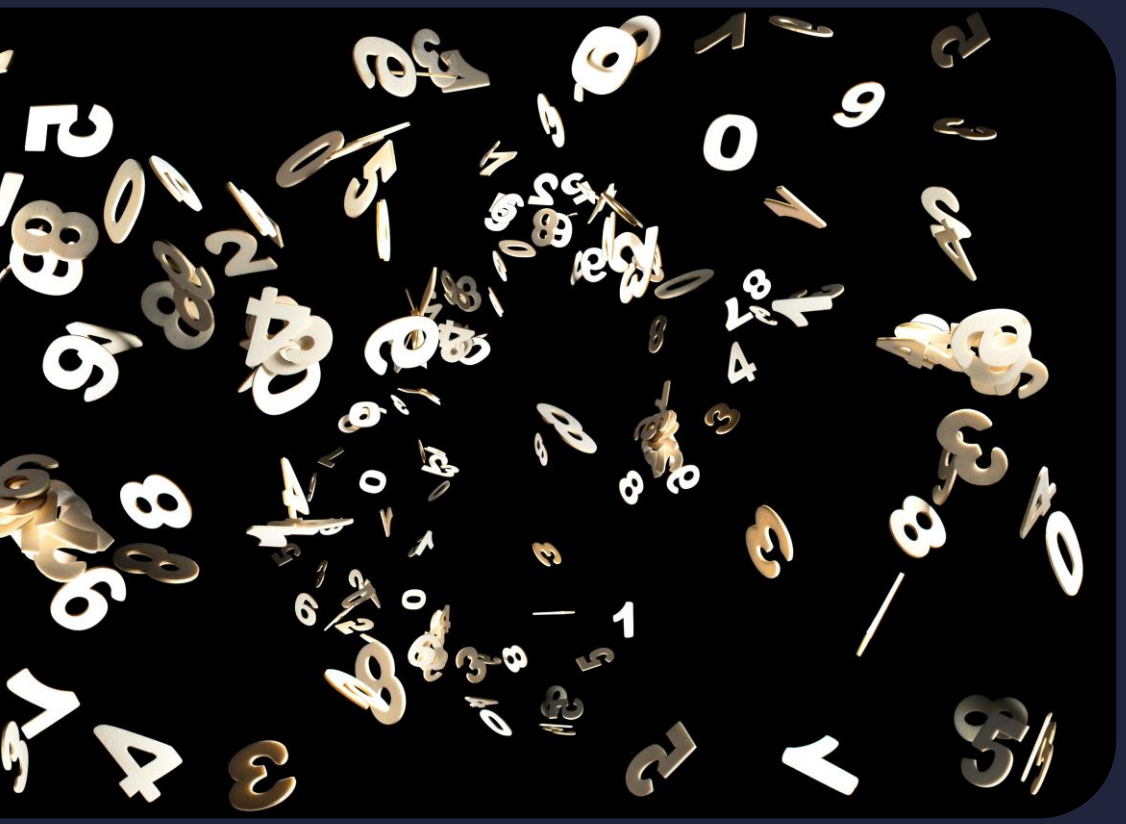
To stop must no longer satisfy condition



Syntax of while loop

```
while (<boolean_expression>)
{
 block of code;
};
```

example



While loop example

To print a list of all numbers that come before the number 100…

- Print a variable
- Add 1 to it
- Print again until all numbers printed

While loop example

```
int num=0;
while(num<100){
printf("%d, ", num);
num++;
};
```

Example

```
0, 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, 3
0, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57,
58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85,
 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99,
--------------------------------
Process exited after 0.09497 seconds with return value 0
Press any key to continue . . .
```

Example

While loop example

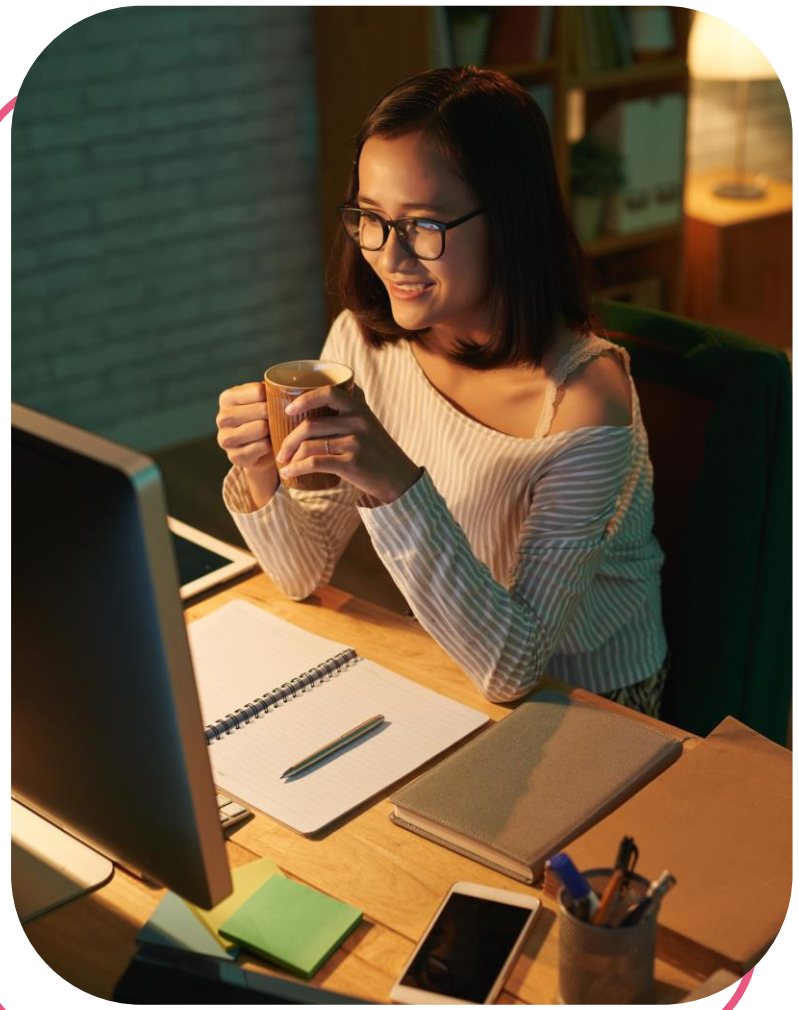
```
int num=0;
while(num<100){
printf("%d, ", num);
num++;
};
```

Example

```
0, 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, 3
0, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57,
58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85,
 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99,
--------------------------------
Process exited after 0.09497 seconds with return value 0
Press any key to continue . . .
```

Example

Do-while loop



Computer first executes code then checks if condition is true

Used to modify condition within the loop first

Used when condition requires user input

Do-while loop example

To read input from user until desired value is entered…

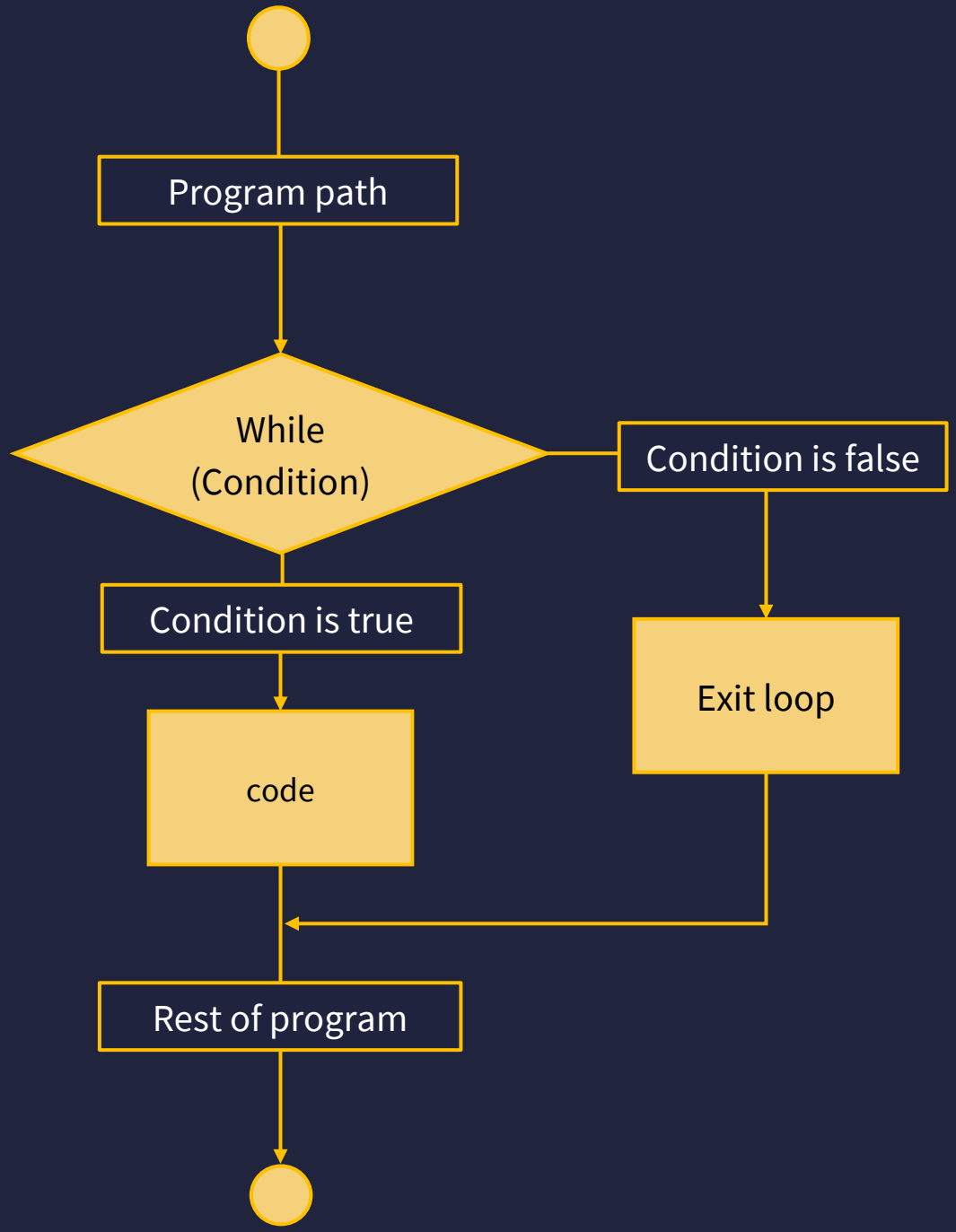
- User enters values
- Computer evaluates input until value makes condition false
- Exists loop

```
int num=0;
  do{
  printf("enter a single digit number
  to exit the loop\n");
       scanf("%d", &num);
       system("cls");
  }
  while(num>10);
printf("%d is a single digit number.
exited loop\n",num);
```

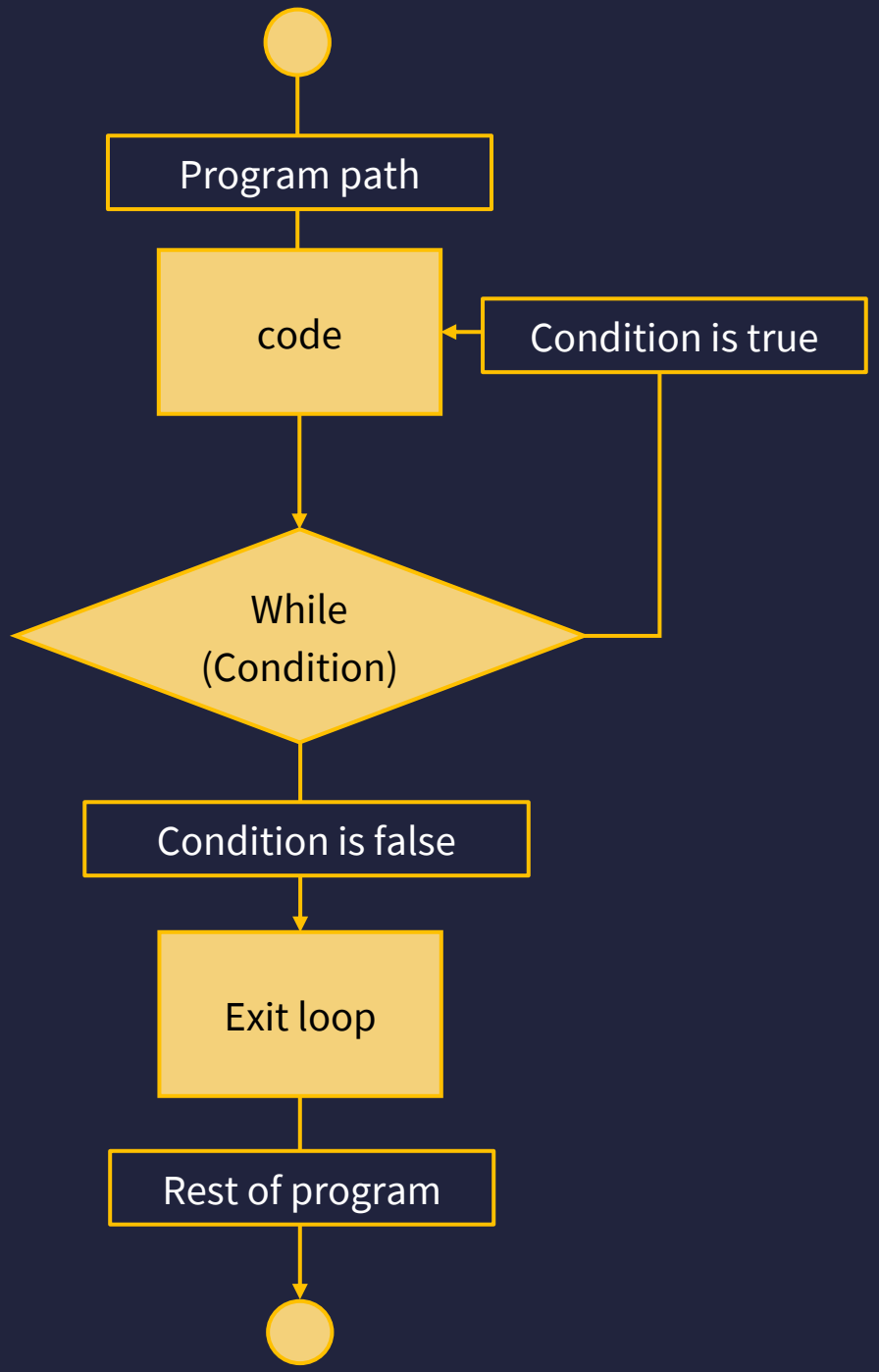
Example

Block of code is executed first before condition is evaluated

While loop flowchart



Do-while loop flowchart



Things to note when using while loops...

Unreachable loop

```
int num=0;
while(num>100){
printf("%d, ", num);
num++;
};
```

Example

Always make sure your condition is satisfiable.

Loop cannot be exited

```
int num=0;
while(num>100){
printf("%d, ", num);
Num--;
};
```

Example

Make sure your condition eventually becomes false.

Did you know?

Infinite loops aren’t always a bad thing.

Iteration and initialising variables...

If you leave the variable uninitialised, when your program is run, the variable is initialised and takes on an unknown value that was already at that memory location.

Note that..

- The variables used in the loop condition must be initialised when the while loop is first encountered.
- Test the loop control variables in the condition of the loop.

For loops

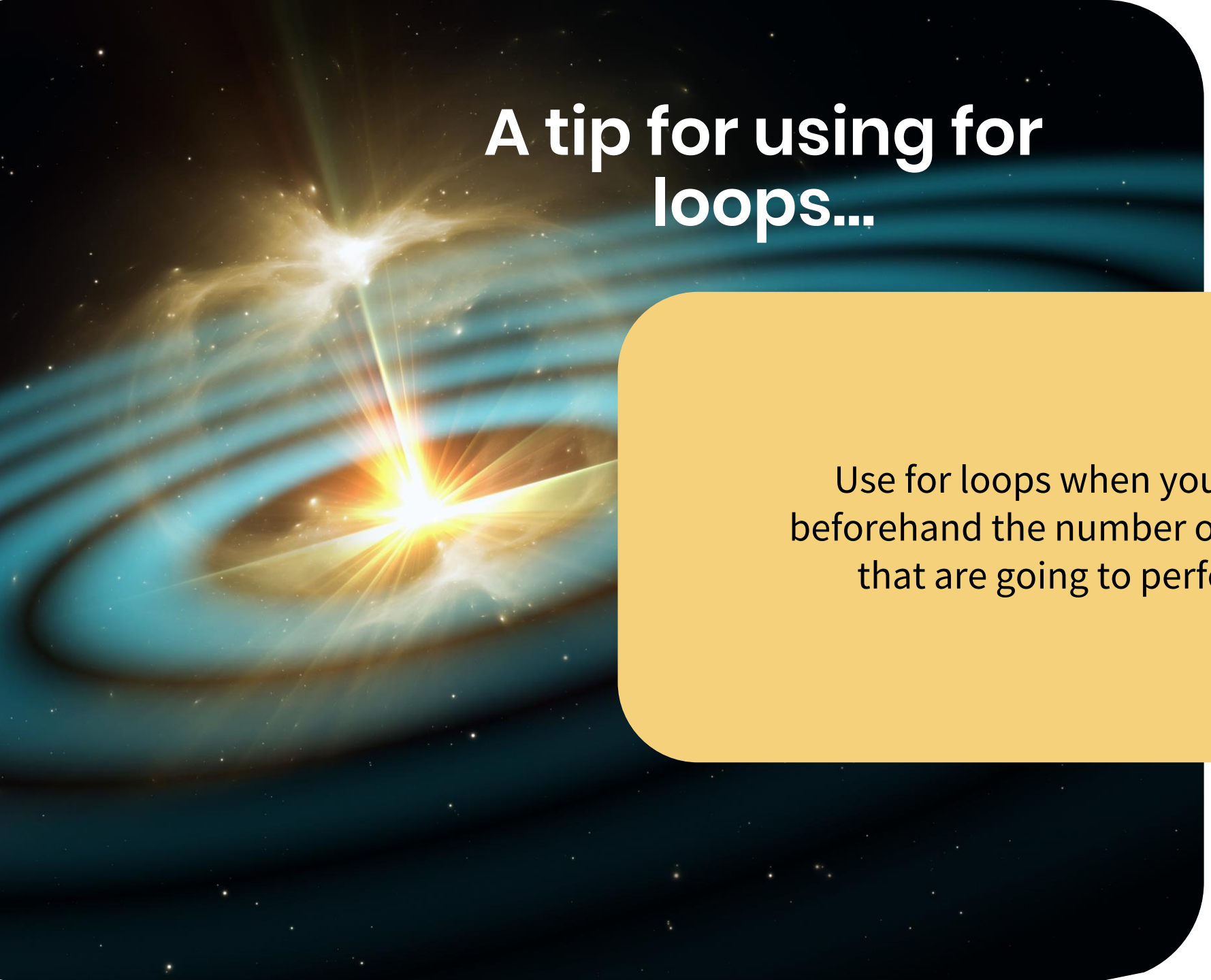
What's the difference?

The **while loop** is more suited for when the actual number of iterations is not clear.

The **for loop** lets you keep track of the number of iterations by using the control variable as a counter.

For loop

- Has an initial condition, final condition, and modification for the variable
- Commonly used for enumerated repetition
- Difficult to run into an infinite loop due to controls



A tip for using for loops...

Use for loops when you can tell beforehand the number of iterations that are going to performed.

Syntax of for loop

Initialisation step before loop begins

Condition tested at beginning of each iteration loop

If true, then body of the loop executed

If false, then block of code within the for loop not executed

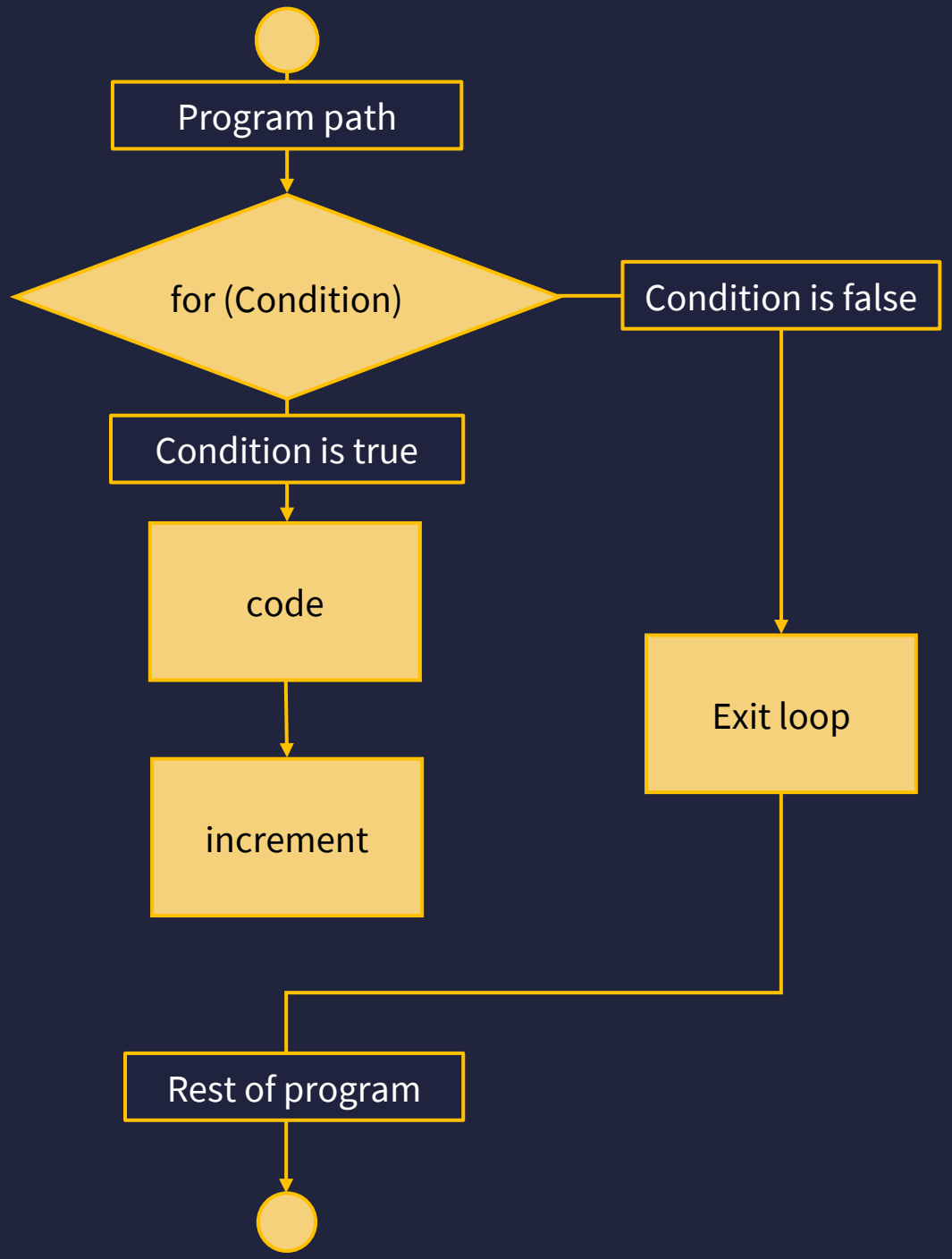
Execution continues with code after the loop

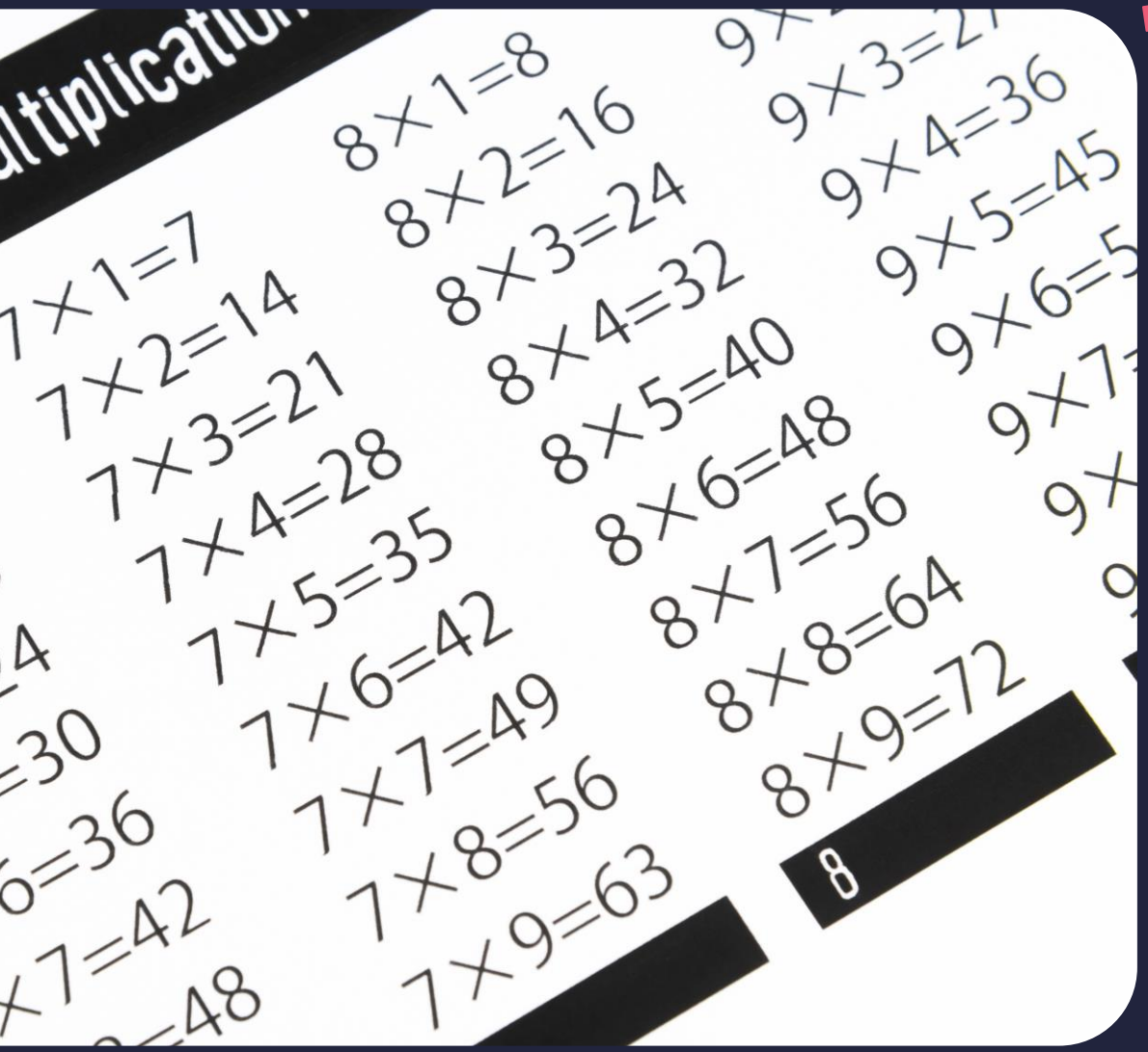
Syntax of for statement

```
for ( init; condition;
increment ) {
    code;
}
```

Example

For loop flowchart





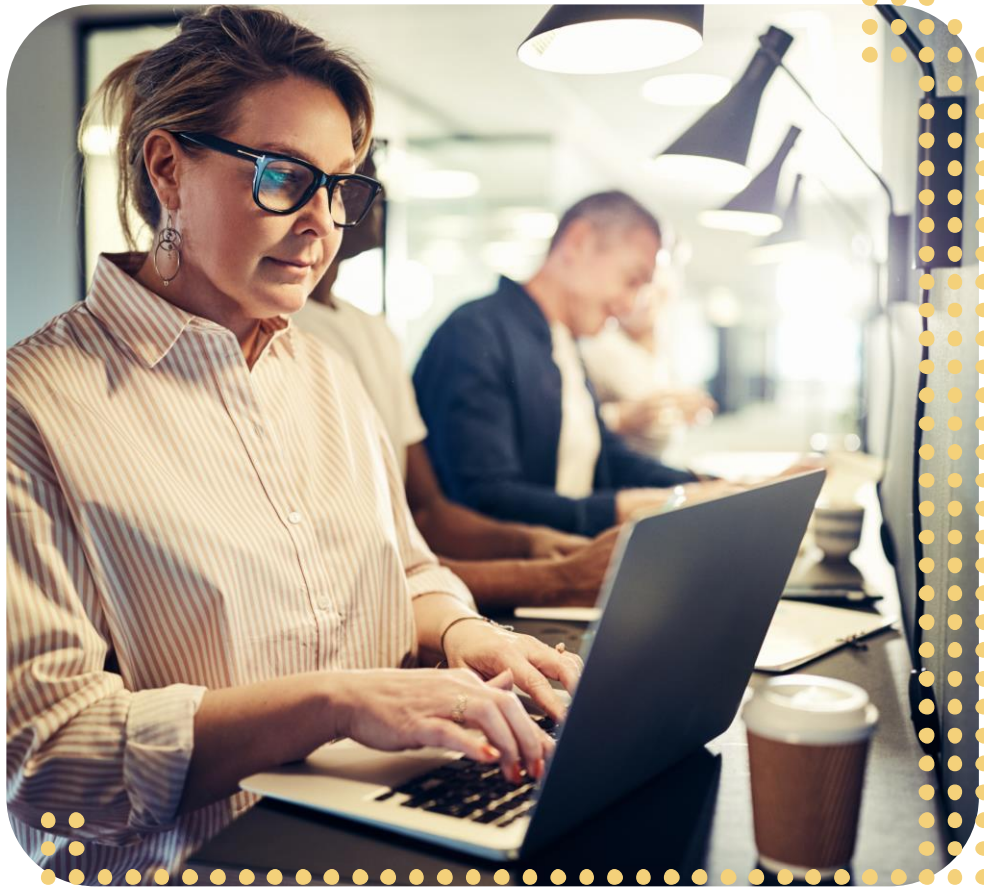
For loop example

Create a program that prints the multiplication tables of any number obtained from the user.

For loop example

```
int number, multiple;
printf("Enter a number: \n");
scanf("%d",&number);
printf("The %d multiplication table is:",number);
for (int i = 1; i <= 12; i++)
{
multiple = number * i;
printf(" %d   ",multiple);
```

Example



Things to note when using for loops...

Possibility of an unreachable loop or an infinite loop is low when working with for loops

Emphasis on counting

Example: Your loop has to take exactly 10 iterations. If you set your initial condition as 1 and your execution as <10. It means there will be only 9 iterations instead of 10.

Recursion

Recursion

A function that calls itself repeatedly

C emphasises iteration over recursion

Two components

General case: solution expressed as a smaller version of itself

Base case: solution stated non-recursively

Factorial: product of an integer and all the integers below it

First five steps are recursive

5! =5*4*3*2*1

We can regroup this so that it reads
5*(4!)

Then again 5*(4*(3!))

And again 5*(4*(3*(2!)))

And again 5*(4*(3*(2*(1!))))

And again 5*(4*(3*(2*(1*(0!)))))

And again 5*(4*(3*(2*(1*(1)))))

Example

The final step isn't recursive

n! = {

1 if n = 0 which is the base case, and

n * (n - 1)! if n > 0 which is the recursive step

Example

Recursion

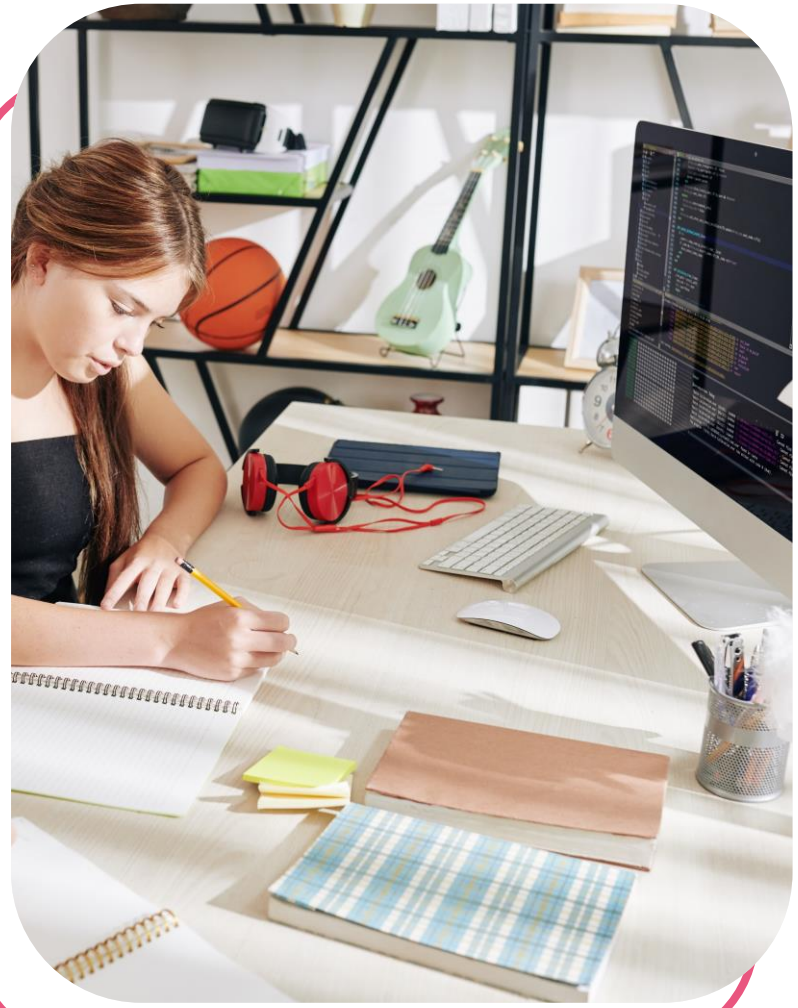
```
long factorial (int n)
{
  long result-0;

  if ( n == 0 )
    result = 1;
  else
    result = n * factorial
  (n-1);
  return result;
}
```

Example

Factorial(5)
↓ recursive step
5 * factorial (4)
↓ recursive step
4 * factorial (3)
↓ recursive step
3 * factorial (2)
↓ recursive step
2 * factorial (1)
↓ recursive step
1 * factorial (0)
↓ base case 1

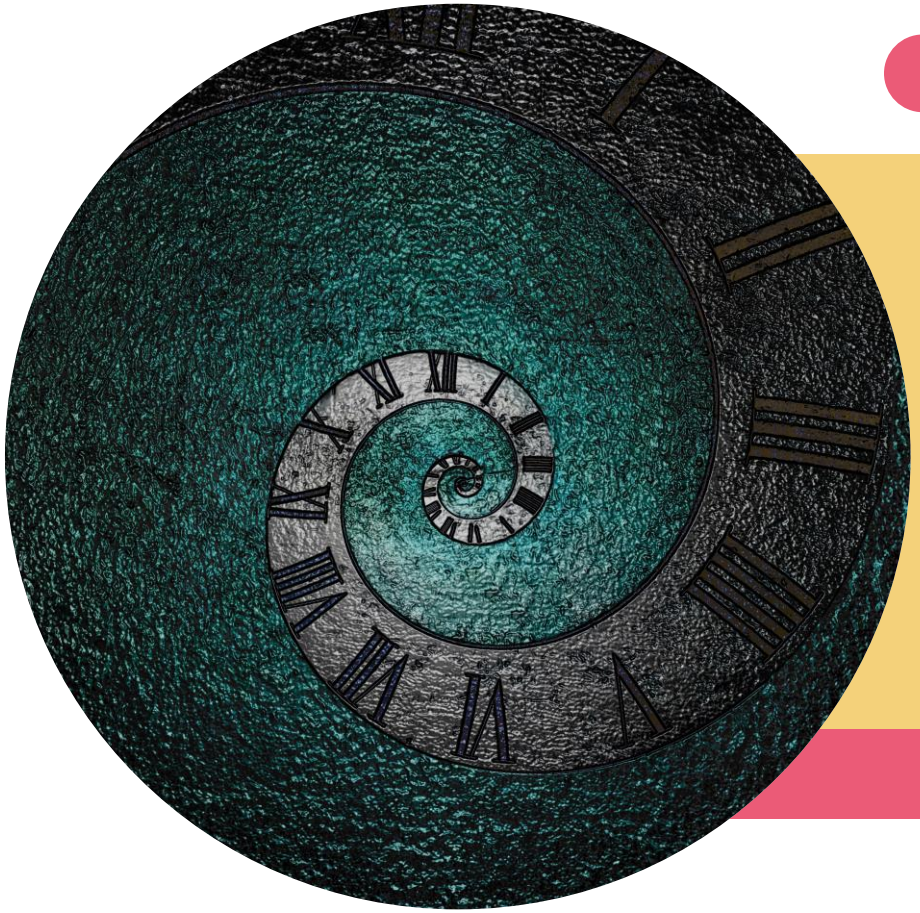
Last step returns the value 120

Why do we need recursions?



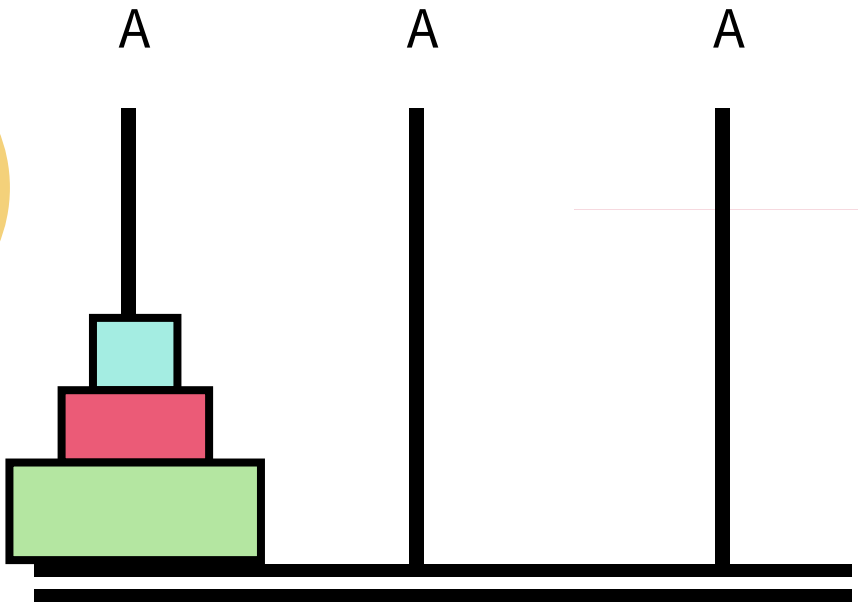
Mathematics

- Exponents and factorials
- Intuitive way of solving problems: can convert solution to the iterative equivalent when every last bit counts



Mathematics

- Basis of divide and conquer algorithm
- For example: Merge Sort



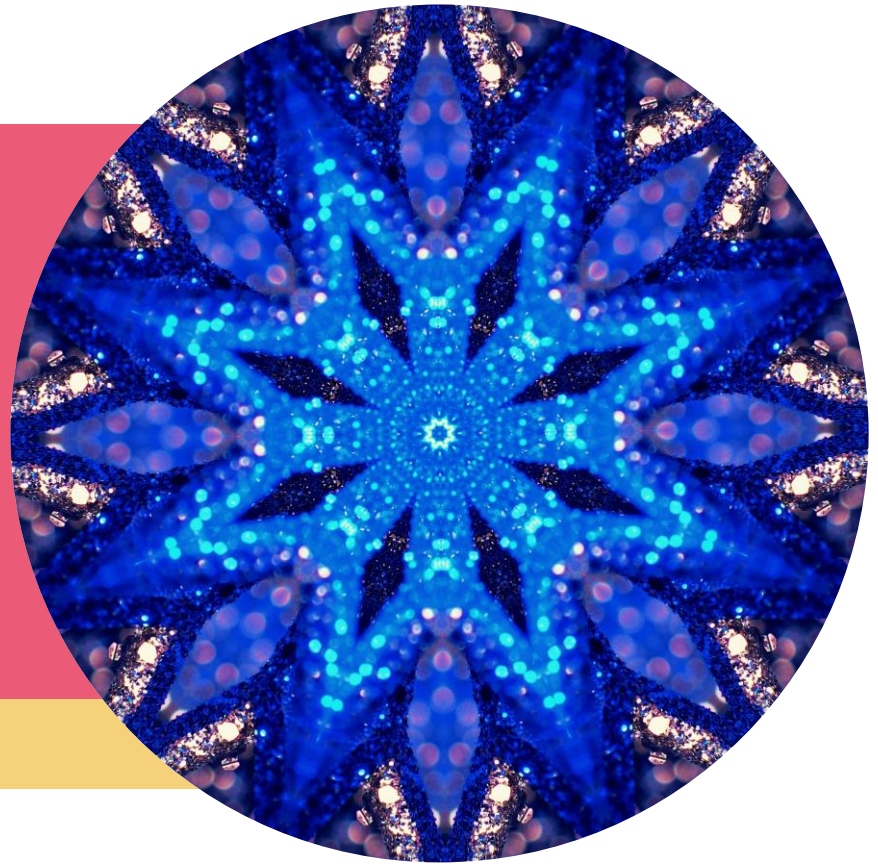
Did you know?

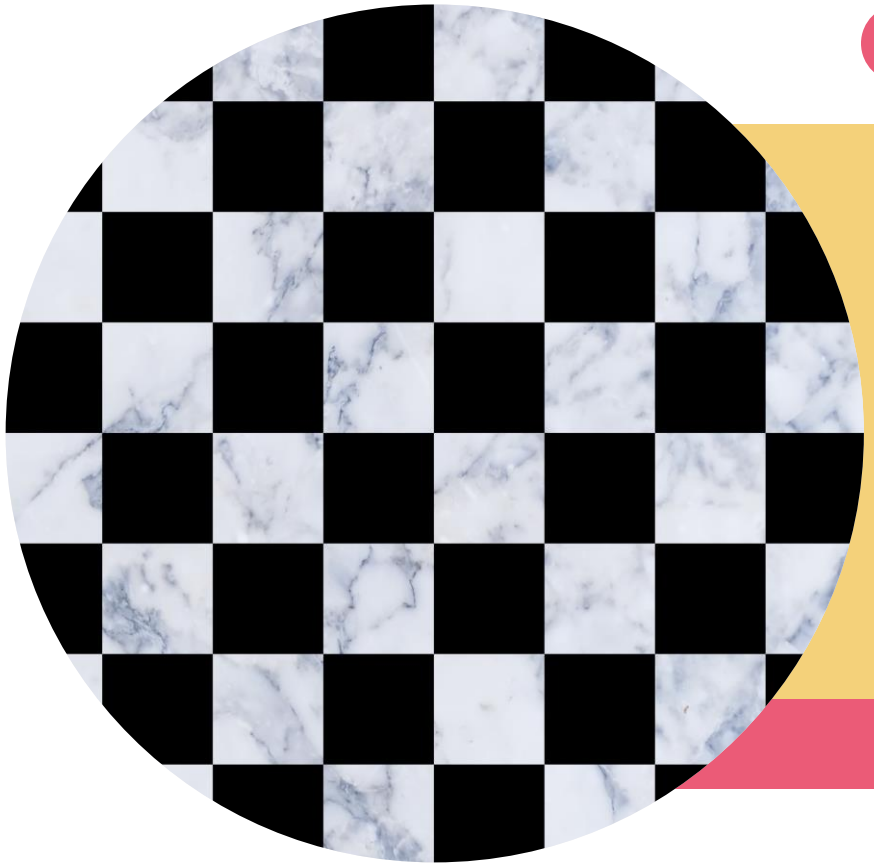
The Towers of Hanoi is a mathematical puzzle with a solution that illustrates the power and elegance of recursion.

Computer graphics

For example:

Fractals and Koch’s snowflake





Artificial intelligence

For example:

- Eight queen’s puzzle
- Maze generation

Things to note when using recursion...

Must always be some base or trivial case that can be solved without recursion

Recursive call must always be to a case that makes progress toward the base case

Use proof by induction

Don’t duplicate work by solving the same instance of a problem in separate recursive calls

Fibonacci sequence

```c
#include<stdio.h>
int Fibonacci(int);
int main()
{
   int n, i = 0, c;

   scanf("%d",&n);
   printf("Fibonacci series\n");
   for ( c = 1 ; c <= n ; c++ )
   {
      printf("%d\n", Fibonacci(i));
      i++;
   }
   return 0;
}
int Fibonacci(int n)
{
   if ( n == 0 )
      return 0;
   else if ( n == 1 )
      return 1;
   else
      return ( Fibonacci(n-1) + Fibonacci(n-2) );
}
```

Example