```
def fib(n):
    if n <= 1:
        return n
    return fib(n-1) + fib(n-2)

print(fib(10))
```

```
function fib(n) {
  if (n <= 1) return n;
  return fib(n - 1) + fib(n - 2);
}
console.log(fib(10));
```

```
public class Main {
  static int fib(int n) {
    if (n <= 1) return n;
    return fib(n-1) + fib(n-2);
  }
  public static void main(String[] args) {
    System.out.println(fib(10));
  }
}
```

```c
#include <stdio.h>
int fib(int n) {
  if (n <= 1) return n;
  return fib(n-1) + fib(n-2);
}
int main() {
  printf("%d\n", fib(10));
  return 0;
}
```

```
#include <iostream>
using namespace std;
int fib(int n) {
  if (n <= 1) return n;
  return fib(n-1) + fib(n-2);
}
int main() {
  cout << fib(10) << endl;
}
```

```
using System;
class Program {
  static int Fib(int n) => n <= 1 ? n : Fib(n-1) +
Fib(n-2);
  static void Main() => Console.WriteLine(Fib(10));
}
```

```
package main
import "fmt"
func fib(n int) int {
  if n <= 1 {
    return n
  }
  return fib(n-1) + fib(n-2)
}
func main() {
  fmt.Println(fib(10))
}
```

```
func fib(_ n: Int) -> Int {
    if n <= 1 { return n }
    return fib(n-1) + fib(n-2)
}
print(fib(10))
```

```
fun fib(n: Int): Int = if (n <= 1) n else fib(n-1) +
fib(n-2)

fun main() {
    println(fib(10))
}
```

```
def fib(n)
  return n if n <= 1
  fib(n-1) + fib(n-2)
end

puts fib(10)
```

```php
<?php
function fib($n) {
  if ($n <= 1) return $n;
  return fib($n-1) + fib($n-2);
}
echo fib(10);
?>
```

```
fib <- function(n) {
  if (n <= 1) return(n)
  fib(n-1) + fib(n-2)
}
cat(fib(10), "\n")
```

```
sub fib {
  my $n = shift;
  return $n if $n <= 1;
  return fib($n-1) + fib($n-2);
}
print fib(10), "\n";
```

```
object Main extends App {
  def fib(n: Int): Int = if (n <= 1) n else fib(n-1)
+ fib(n-2)
  println(fib(10))
}
```

```
defmodule Fib do
  def calc(0), do: 0
  def calc(1), do: 1
  def calc(n), do: calc(n-1) + calc(n-2)
end

IO.puts(Fib.calc(10))
```

```
fn fib(n: u32) -> u32 {
    match n {
        0 | 1 => n,
        _ => fib(n - 1) + fib(n - 2),
    }
}
fn main() {
    println!("{}", fib(10));
}
```

```
function fib(n)
    n <= 1 && return n
    fib(n-1) + fib(n-2)
end
println(fib(10))
```

```
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
main = print (fib 10)
```

```
function fib(n)
  if n <= 1 then return n end
  return fib(n-1) + fib(n-2)
end
print(fib(10))
```

```
int fib(int n) => n <= 1 ? n : fib(n-1) + fib(n-2);
void main() {
  print(fib(10));
}
```

```
#import <Foundation/Foundation.h>
int fib(int n){return n<=1?n:fib(n-1)+fib(n-2);}
int
main(){@autoreleasepool{NSLog(@"%d",fib(10));}return
0;}
```

```
-module(fib).
-export([calc/1]).
calc(0) -> 0;
calc(1) -> 1;
calc(N) -> calc(N-1) + calc(N-2).
```

```
let rec fib n = if n <= 1 then n else fib(n-1) +
fib(n-2)
[<EntryPoint>]
let main _ =
  printfn "%d" (fib 10)
  0
```

```
(define (fib n)
  (if (<= n 1)
      n
      (+ (fib (- n 1)) (fib (- n 2)))))
(display (fib 10)) (newline)
```

```
(defun fib (n)
  (if (<= n 1) n (+ (fib (- n 1)) (fib (- n 2)))))
(print (fib 10))
```

```
fib(0, 0).
fib(1, 1).
fib(N, F) :-
  N > 1,
  N1 is N - 1,
  N2 is N - 2,
  fib(N1, F1), fib(N2, F2),
  F is F1 + F2.
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID. FIB.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 N PIC 9(2) VALUE 10.
PROCEDURE DIVISION.
DISPLAY FUNCTION FIBONACCI(N).
STOP RUN.
```

```
with Ada.Text_IO; use Ada.Text_IO;
function Fib(N : Integer) return Integer is
begin
  if N <= 1 then return N;
  else return Fib(N-1) + Fib(N-2);
  end if;
end Fib;
procedure Main is begin
  Put_Line(Integer'Image(Fib(10)));
end Main;
```

```
fib := [:n | n <= 1 ifTrue:[n] ifFalse:[(fib
value:(n-1)) + (fib value:(n-2))]].
Transcript show: (fib value:10); cr.
```

```
proc fib(n: int): int =
  if n <= 1: n
  else: fib(n-1) + fib(n-2)
echo fib(10)
```

```
recursive function fib(n) result(f)
  integer :: n, f
  if (n <= 1) then
    f = n
  else
    f = fib(n-1) + fib(n-2)
  end if
end function fib
print *, fib(10)
```

```
(define (fib n)
  (if (<= n 1)
      n
      (+ (fib (- n 1)) (fib (- n 2)))))
(display (fib 10))(newline)
```

```
let rec fib n = if n <= 1 then n else fib (n-1) +
fib (n-2)
let () = print_int (fib 10)
```

```
program Fib;
function F(n: integer): integer;
begin
  if n <= 1 then F := n
  else F := F(n-1) + F(n-2);
end;
begin
  writeln(F(10));
end.
```

```
entity fib is end;
architecture Behavioral of fib is
function fib(n: integer) return integer is
begin
  if n <= 1 then return n;
  else return fib(n-1) + fib(n-2);
  end if;
end;
begin
end;
```

```
def fib(n) { n <= 1 ? n : fib(n-1) + fib(n-2) }
println fib(10)
```

```
proc fib {n} {
  if {$n <= 1} {return $n}
  expr {[fib [expr {$n-1}]] + [fib [expr {$n-2}]]}
}
puts [fib 10]
```

```
+[--->++<]>+.++++++++++.+++++++..+++.[->+++++<]>+.-
----------.++++++++++++.----------.-[--->+<]>---.++
+.------.--------.
```