

Assignment15

Problem Statement

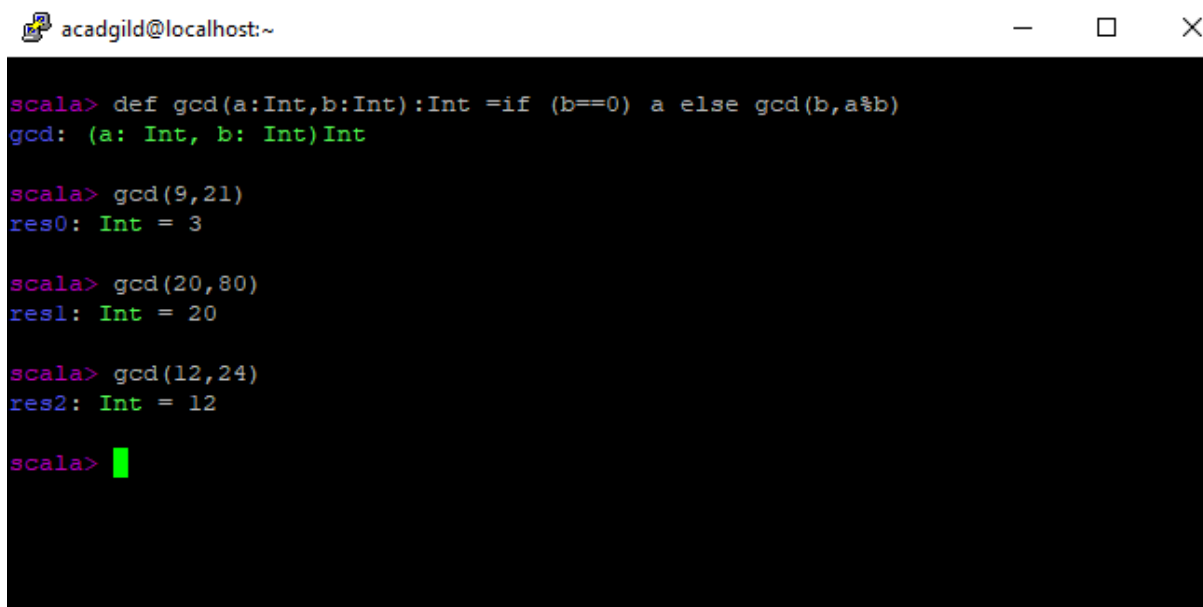
Task 1

Create a Scala application to find the GCD of two numbers

➔ Command : `def gcd(a:Int,b:Int):Int =if (b==0) a else gcd(b,a%b)`

To get the output

➔ Command : `gcd(9,21)`

A screenshot of a Scala REPL window. The window title is 'acadgild@localhost:~'. The background is black with text in red, green, and white. The code entered is: `scala> def gcd(a:Int,b:Int):Int =if (b==0) a else gcd(b,a%b)`, followed by `gcd: (a: Int, b: Int)Int`. Then, `scala> gcd(9,21)` is entered, resulting in `res0: Int = 3`. Next, `scala> gcd(20,80)` is entered, resulting in `res1: Int = 20`. Then, `scala> gcd(12,24)` is entered, resulting in `res2: Int = 12`. Finally, `scala>` is entered with a green cursor. The window has standard OS controls (minimize, maximize, close) in the top right corner.

```
acadgild@localhost:~  
scala> def gcd(a:Int,b:Int):Int =if (b==0) a else gcd(b,a%b)  
gcd: (a: Int, b: Int)Int  
  
scala> gcd(9,21)  
res0: Int = 3  
  
scala> gcd(20,80)  
res1: Int = 20  
  
scala> gcd(12,24)  
res2: Int = 12  
  
scala> █
```

Task 2

Fibonacci series (starting from 1) written in order without any spaces in between, thus producing a sequence of digits.

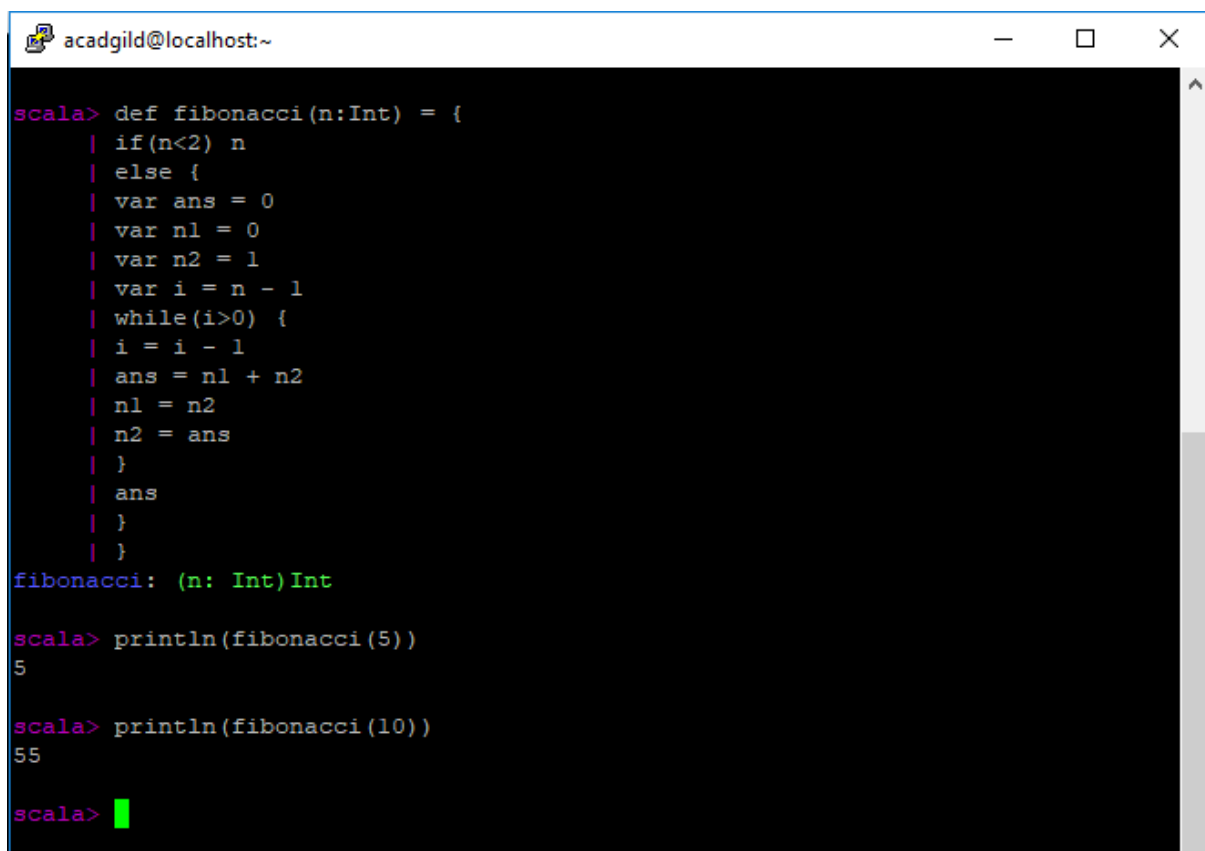
Write a Scala application to find the Nth digit in the sequence.

➤ Write the function using standard for loop

➔ Command: `def Fibonacci (n:Int) = {`
 `if(n<2) n`
 `Else {`
 `Var ans = 0`
 `Var n1 = 0`
 `Var n2 = 1`
 `Var i = n - 1`
 `While(i>0) {`
 `i = i - 1`
 `ans = n1 + n2`
 `n1 = n2`
 `n2 = ans`
 `}`
 `ans`
 `}`
}

To print the output

➔ Command : `println(fibonacci(5))`



```
acadgild@localhost:~  
scala> def fibonacci(n:Int) = {  
  | if(n<2) n  
  | else {  
  |   var ans = 0  
  |   var n1 = 0  
  |   var n2 = 1  
  |   var i = n - 1  
  |   while(i>0) {  
  |     i = i - 1  
  |     ans = n1 + n2  
  |     n1 = n2  
  |     n2 = ans  
  |   }  
  |   ans  
  | }  
fibonacci: (n: Int)Int  
scala> println(fibonacci(5))  
5  
scala> println(fibonacci(10))  
55  
scala> 
```

➤ Write the function using recursion

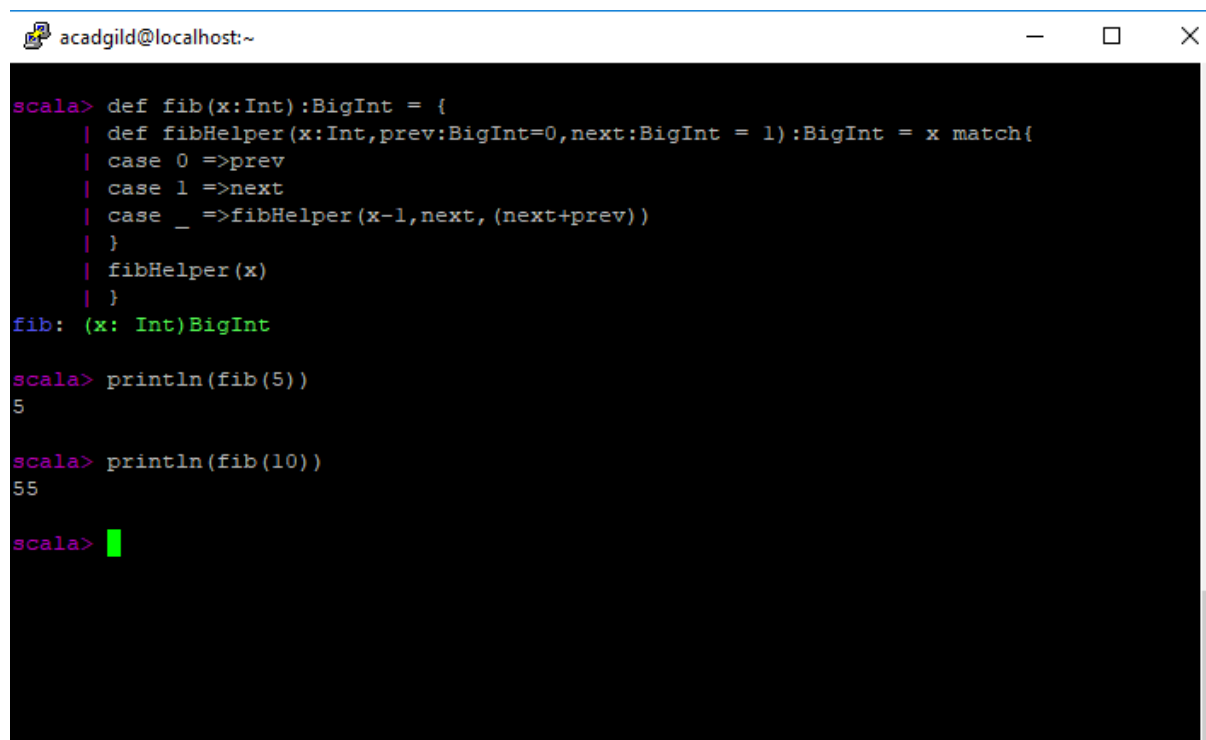
➔ Command :

```
def fib(x:Int):BigInt = {  
  Def fibHelper(x:Int,prev:BigInt=0,next:BigInt = 1):BigInt = x match{  
    case 0 =>prev  
    case 1 =>next  
    case _ =>fibHelper(x-1,next,(next+prev))  
  }  
  fibHelper(x)  
}
```

To print the output

➔ Command :

```
println(fib(5))
```



```
acadgild@localhost:~  
scala> def fib(x:Int):BigInt = {  
  | def fibHelper(x:Int,prev:BigInt=0,next:BigInt = 1):BigInt = x match{  
  | case 0 =>prev  
  | case 1 =>next  
  | case _ =>fibHelper(x-1,next,(next+prev))  
  | }  
  | fibHelper(x)  
  | }  
fib: (x: Int)BigInt  
scala> println(fib(5))  
5  
scala> println(fib(10))  
55  
scala> 
```

Task 3

Find square root of number using Babylonian method.

1. Start with an arbitrary positive start value x (the closer to the root, the better).
2. Initialize $y = 1$.
3. Do following until desired approximation is achieved.
 - a) Get the next approximation for root using average of x and y
 - b) Set $y = n/x$

```
➔ Command : def sqrt(a:Double) = {  
    Val acc = 1e-10  
    Def findRoot(x:Double):Double = {  
        Val nextx = (a/x + x)/2  
        If((x-nextx).abs<acc*x) nextx else findRoot(nextx)  
    }  
    findRoot(1)  
}
```

To print the Output

```
➔ Command : println(sqrt(121))
```

acadgild@localhost:~

— □ ×

```
scala> def sqrt(a:Double) = {  
  | val acc = 1e-10  
  | def findRoot(x:Double):Double = {  
  | val nextx = (a/x + x)/2  
  | if((x-nextx).abs<acc*x) nextx else findRoot(nextx)  
  | }  
  | findRoot(1)  
  | }  
sqrt: (a: Double)Double  
  
scala> println(sqrt(121))  
11.0  
  
scala> println(sqrt(81))  
9.0  
  
scala> █
```