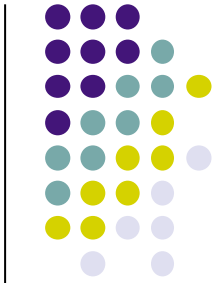
Recursion and loops

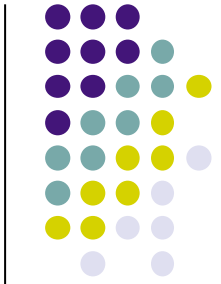


- In the previous lesson we saw SumDigitsR:

```
fun {SumDigitsR N}
      if (N==0) then 0
      else (N mod 10) + {SumDigitsR (N div 10)} end
end
```

- The recursive call and the condition together act like a loop: a calculation that is repeated to achieve a result
  - Each execution of the function body is one iteration of the loop
- Recursion can be used to make a loop
  - In this lesson we will go to the root of this intuition

Invariant programming



- A loop is a part of a program that is repeated until a condition is satisfied
  - Loops are an important technique in all paradigms
  - Loops are a special case of recursion, called *tail recursion*, where the recursive call is the last operation done in the function body
- We will give a general technique, invariant programming, to program correct and efficient loops
  - Loops are often very difficult to get exactly right, and invariant programming is an excellent way to achieve this
  - This applies to *both* declarative and imperative paradigms
- New concepts introduced in this lesson
  - Specification, accumulator, principle of communicating vases