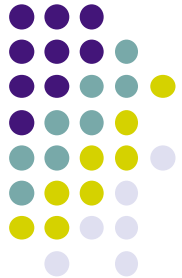
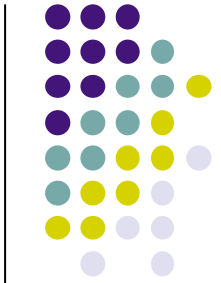
Why do we need semantics?



- If you do not understand something, then you do not master it – it masters you!
  - If you know nothing about how a car works, then a car mechanic can charge you whatever he wants
  - If you do not understand how government works, then you cannot vote wisely and the government becomes a tyranny
- The same holds true for programming
  - To write correct programs and to understand other people's programs, you have to understand the language deeply
  - All software developers should have this level of understanding
  - The goal of this lesson is to show you how

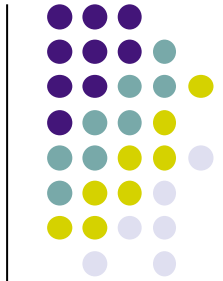
What is the semantics of a language?



- The semantics of a programming language, also called formal semantics or mathematical semantics, is a completely precise explanation of how programs execute that can be used to reason about program design and correctness
- We will give a semantics for all the paradigms of this course
  - We start by giving the semantics of the functional paradigm
- We have already seen the first part, namely the *kernel language*
  - In this lesson we will see the second part, namely the *abstract machine*

- Before taking the plunge into the abstract machine, let's take a step back and talk about semantics in general

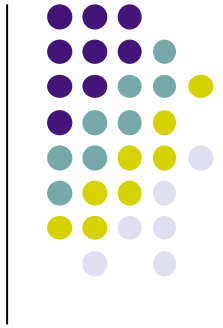
How can we define language semantics?



- Four general approaches have been invented:
  - Operational semantics: Explains a program in terms of its execution on a rigorously defined abstract machine
  - Axiomatic semantics: Explains a program as an implication: if certain properties hold before the execution, then some other properties will hold after the execution
  - Denotational semantics: Explains a program as a function over an abstract domain, which simplifies certain kinds of mathematical analysis of the program
  - Logical semantics: Explains a program as a logical model of a set of logical axioms, so program execution is deduction: the result of a program is a true property derived from the axioms

- The operational semantics works for all paradigms (since all programs run on computers!)
  - The other approaches are less general; they work best for some paradigms
  - To reason about correctness, we will complement the operational semantics with ideas taken from the other approaches

Operational semantics



- The operational semantics has two parts
  - Kernel language: first, translate the program into the kernel language
  - Abstract machine: then, execute the program on the abstract machine

- We will introduce the operational semantics with an example that uses it to prove correctness of a program
  - After this introduction, we will define the abstract machine and give an example of how it executes
  - Then we will define the semantic rules for each instruction of the kernel language
  - Finally, we will take a special look at procedure definition and call, since they are very important (higher-order programming and data abstraction)