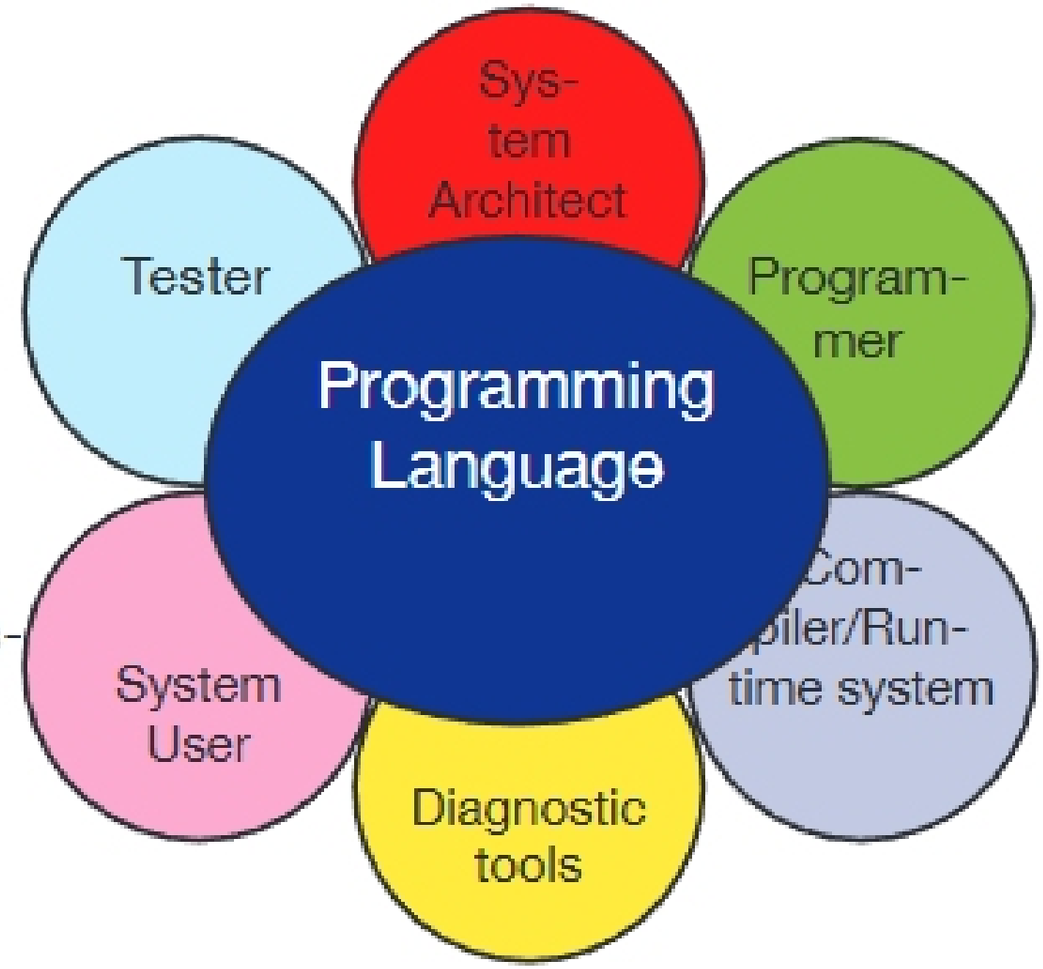
Types

- General discussion of types
  - What is a type?
  - Compile-time vs run-time checking
  - Conservative program analysis
- Type inference
  - Good example of static analysis algorithm
  - Will study algorithm and examples
- Polymorphism
  - Polymorphism vs overloading
  - Uniform vs non-uniform implementation of polymorphism

Language goals and tradeoffs

Thoughts to keep in mind

- What features are convenient for programmer?
- What other features do type systems prevent?
- What are design tradeoffs?
  - Easy to write but harder to read?
  - Easy to write but poorer error messages?
- What are the implementation costs?



Type

A type is a collection of computable values that share some structural property and/or operations.

- Examples
  - Integers
  - Strings
  - int -> bool
  - (int * int) -> bool
- "Non-examples"
  - { 3, true, fun x = x }
  - Even integers
  - { f:int -> int | f(x+1) > f(x) }

Distinction between sets that are types and sets that are not types is language dependent--for example:

– no string type in C v.s. String in ML

– char in C v.s. no character type in ML

– different numeric types: short, long, real, float, int, etc.