What is a pattern?

- **"... a three part rule, which expresses a relation between a certain context, a problem, and a solution. The pattern is, in short, at the same time a thing, ... , and the rule which tells us how to create that thing, and when we must create it."**

  Christopher Alexander

name	*factory, aka virtual constructor*
problem	*delegate creation responsibility: Hyperwag*
solution	*createFoo() method returns aFoo, bFoo,...*
consequences	*potentially lots of subclassing, ...*

- **more a recipe than a plan, micro-architecture, frameworks, language idioms made abstract, less than a principle but more than a heuristic**
- **patterns capture important practice in a form that makes the practice accessible**

Patterns are discovered, not invented

- You encounter the same "pattern" in developing solutions to programming or design problems
  - develop the pattern into an appropriate form that makes it accessible to others
  - fit the pattern into a language of other, related patterns
- Patterns transcend programming languages, but not (always) programming paradigms
  - OO folk started the patterns movement
  - language idioms, programming templates, programming patterns, case studies

Pattern/Programming Interlude

- Microsoft interview question (1998)

3	3	5	5	7	8	8	8
3	5	7	8				

- Dutch National Flag problem (1976)

🟥	⬜	🟦	🟦	🟥	⬜	🟦	⬜	🟥
🟥	🟥	🟥	⬜	⬜	⬜	🟦	🟦	🟦

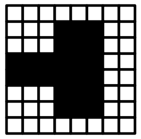
- Remove Zeros (AP 1987)

2	1	0	5	0	0	8	4
2	1	5	8	4			

- Quicksort partition (1961, 1986)

4	3	8	9	1	6	0	5
3	1	0	4	8	9	6	5

- Run-length encoding (SIGCSE 1998)



```
11 3 5 3 2 6 2 6 5 3 5 3 5 3 10
```

One loop for linear structures

- ✍ Algorithmically, a problem may seem to call for multiple loops to match intuition on how control structures are used to program a solution to the problem, but data is stored sequentially, e.g., in an array or file. Programming based on control leads to more problems than programming based on structure.

  Therefore, use the structure of the data to guide the programmed solution: one loop for sequential data with appropriately guarded conditionals to implement the control

  Consequences: one loop really means loop according to structure, do not add loops for control: what does the code look like for run-length encoding example?

Coding Pattern

- **Name:**
  - one loop for linear structures
- **Problem:**
  - Sequential data, e.g., in an array or a file, must be processed to perform some algorithmic task. At first it may seem that multiple (nested) loops are needed, but developing such loops correctly is often hard in practice.
- **Solution:**
  - Let the structure of the data guide the coding solution. Use one loop with guarded/if statements when processing one-dimensional, linear/sequential data
- **Consequences:**
  - Code is simpler to reason about, facilitates develop of loop invariants, possibly leads to (slightly?) less efficient code

Design patterns you shouldn't miss

- **Iterator**
  - **useful in many contexts, see previous examples, integral to both C++ and Java**
- **Factory**
  - **essential for developing OO programs/classes, e.g., create iterator from a Java 1.2 List?** `list.iterator()`
- **Composite**
  - **essential in GUI/Widget programming, widgets contain collections of other widgets**
- **Command**
  - **encapsulate a request as an object, supports undo, re-usable commands (compare anonymous inner class)**
- **Observer/Observable, Publish/Subscribe, MVC**
  - **separate the model from the view, smart updates**

More Design Patterns

- Singleton
  - a class has a single instance, enforce this via design rather than convention
- Adapter/Façade
  - replug-and-play, hide details
- Mediator
  - define a class that encapsulates how other objects interact, promote loose coupling since other objects interact with mediator instead of with each other: Gui, App, Controller
- Proxy
  - provide a surrogate/placeholder for ease of use, different control: smart stack-based pointer for iterators in C++

CPS 108

- The software process
  - how to make good programs, how to deliver good products
  - know what questions to ask, scenarios and small examples
- Working in teams/groups on bigger programs
  - managing teams, working together, using tools like CVS
- Knowledge of multiple programming languages
  - C++, some low level details, what to expect from its C core
  - Java, introduction, where to find useful packages
  - compare/contrast, which language will you turn to?
- On becoming an object-oriented programmer and designer
  - critique your own code, refactor to make simpler based on ability to abstract and generalize
  - use inheritance wisely, know about design patterns