CS 220: Survey of Programming Languages

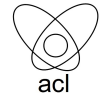
Lecture Slides

Some Properties of Programming Languages

Jan Michael C. Yap

Algorithms and Complexity Laboratory
Department of Computer Science
University of the Philippines, Diliman
jcyap@dcs.upd.edu.ph

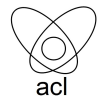
Session 3



Some Properties of Programming Languages

Preliminaries

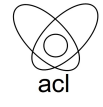
Some Properties of Programming Languages



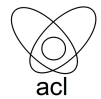
Some Properties of Programming Languages

Preliminaries

Some Properties of Programming Languages

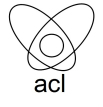


What makes a great programming language?



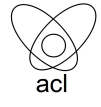
What makes a great programming language?

- There is no standard set of criteria for a great programming language!



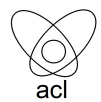
What makes a great programming language?

- There is no standard set of criteria for a great programming language!
  - The set of criteria is more of a suggested list rather than a hardlined one



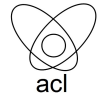
What makes a great programming language?

- There is no standard set of criteria for a great programming language!
  - The set of criteria is more of a suggested list rather than a hardlined one
- Choice of criteria based on need for a PL that can be used to create readable, rewritable, and reliable code.



What makes a great programming language?

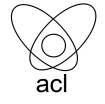
- There is no standard set of criteria for a great programming language!
  - The set of criteria is more of a suggested list rather than a hardlined one
- Choice of criteria based on need for a PL that can be used to create readable, rewritable, and reliable code.
  - The set is taken from R.A. Finkel's and R.W. Sebesta's lists.



Some Properties of Programming Languages

Preliminaries

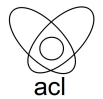
Some Properties of Programming Languages



Programming language properties

	CRITERIA		
Characteristic	READABILITY	WRITABILITY	RELIABILITY
Data types	•	•	•
Syntax design	•	•	•
Support for abstraction		•	•
Expressivity		•	•
Exception handling			•
Restricted aliasing			•



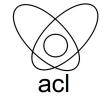



Programming language properties

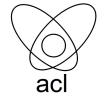
	CRITERIA		
Characteristic	READABILITY	WRITABILITY	RELIABILITY
Data types	•	•	•
Syntax design	•	•	•
Support for abstraction		•	•
Expressivity		•	•
Exception handling			•
Restricted aliasing			•

- Addendum: Efficiency (under reliability)



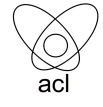



Data types



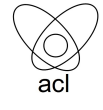
Data types

- The presence of adequate facilities for defining data types and data structures significantly aids readability



Data types

- The presence of adequate facilities for defining data types and data structures significantly aids readability
- Example: explicitly having a Boolean data type



Data types

- The presence of adequate facilities for defining data types and data structures significantly aids readability
- Example: explicitly having a Boolean data type
- Issue though may be on the orthogonality of a PL

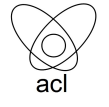


Syntax design

Syntax design

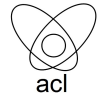
- Closely tied to the concept of clarity
  - Mechanisms should be well defined, and the outcome of code should be easily predictable



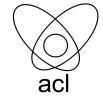
Syntax design

- Closely tied to the concept of clarity
  - Mechanisms should be well defined, and the outcome of code should be easily predictable
- Can also be tied to the concept of a PL's simplicity and orthogonality



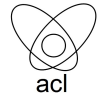
Syntax design

- Closely tied to the concept of clarity
  - Mechanisms should be well defined, and the outcome of code should be easily predictable
- Can also be tied to the concept of a PL's simplicity and orthogonality
  - Simplicity pertains that PLs should have as few basic concepts as possible
  - The more orthogonal the design of a language, the fewer exceptions the language rules require



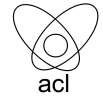
Syntax design

- Closely tied to the concept of clarity
  - Mechanisms should be well defined, and the outcome of code should be easily predictable
- Can also be tied to the concept of a PL's simplicity and orthogonality
  - Simplicity pertains that PLs should have as few basic concepts as possible
  - The more orthogonal the design of a language, the fewer exceptions the language rules require
- Factors to consider in syntax design



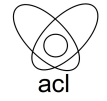
Syntax design

- Closely tied to the concept of clarity
  - Mechanisms should be well defined, and the outcome of code should be easily predictable
- Can also be tied to the concept of a PL's simplicity and orthogonality
  - Simplicity pertains that PLs should have as few basic concepts as possible
  - The more orthogonal the design of a language, the fewer exceptions the language rules require
- Factors to consider in syntax design
  - Special words



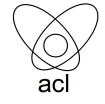
Syntax design

- Closely tied to the concept of clarity
  - Mechanisms should be well defined, and the outcome of code should be easily predictable
- Can also be tied to the concept of a PL's simplicity and orthogonality
  - Simplicity pertains that PLs should have as few basic concepts as possible
  - The more orthogonal the design of a language, the fewer exceptions the language rules require
- Factors to consider in syntax design
  - Special words
  - Form

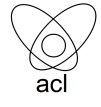


Syntax design

- Closely tied to the concept of clarity
  - Mechanisms should be well defined, and the outcome of code should be easily predictable
- Can also be tied to the concept of a PL's simplicity and orthogonality
  - Simplicity pertains that PLs should have as few basic concepts as possible
  - The more orthogonal the design of a language, the fewer exceptions the language rules require
- Factors to consider in syntax design
  - Special words
  - Form
  - Meaning

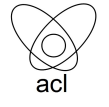


Support for Abstraction



Support for Abstraction

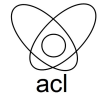
- There should be a way to factor out recurring patterns



Support for Abstraction

- There should be a way to factor out recurring patterns
  - This allows definition and then use complicated structures or operations in ways that allow many of the details to be ignored

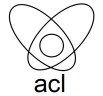


Support for Abstraction

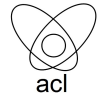
- There should be a way to factor out recurring patterns
  - This allows definition and then use complicated structures or operations in ways that allow many of the details to be ignored
- Related concepts



Support for Abstraction

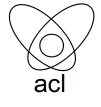
- There should be a way to factor out recurring patterns
  - This allows definition and then use complicated structures or operations in ways that allow many of the details to be ignored
- Related concepts
  - Information hiding



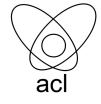
Support for Abstraction

- There should be a way to factor out recurring patterns
  - This allows definition and then use complicated structures or operations in ways that allow many of the details to be ignored
- Related concepts
  - Information hiding
  - Modularity



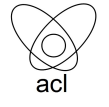
Expressivity



Expressivity

- Expressivity means that a language has relatively convenient, rather than cumbersome, ways of specifying computations.



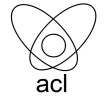



Expressivity

- Expressivity means that a language has relatively convenient, rather than cumbersome, ways of specifying computations.
- Example: Short-circuit operators and the `case` statement

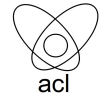


Exception handling



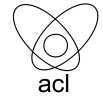
Exception handling

- Exception handling refers to the ability of a program to intercept run-time error/s, take corrective measures, and then continue



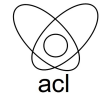
Exception handling

- Exception handling refers to the ability of a program to intercept run-time error/s, take corrective measures, and then continue
- Examples: (Most) Object-oriented languages

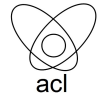


Exception handling

- Exception handling refers to the ability of a program to intercept run-time error/s, take corrective measures, and then continue
- Examples: (Most) Object-oriented languages
- Bad examples: Awk and SNOBOL silently convert data types in order to apply operators

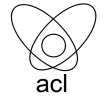


Restricted aliasing



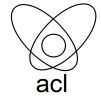
Restricted aliasing

- Aliasing is having two or more distinct names that can be used to access the same memory cell



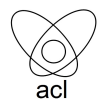
Restricted aliasing

- Aliasing is having two or more distinct names that can be used to access the same memory cell
- It is widely accepted that aliasing is a dangerous feature in a PL

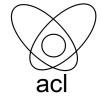


Restricted aliasing

- Aliasing is having two or more distinct names that can be used to access the same memory cell
- It is widely accepted that aliasing is a dangerous feature in a PL
- In some languages, aliasing is used to overcome deficiencies in the language's data abstraction facilities

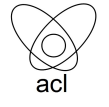


Efficiency



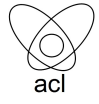
Efficiency

- Efficiency in terms of compiling and executing a program



Efficiency

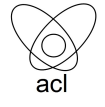
- Efficiency in terms of compiling and executing a program
- Some concepts related to efficiency



Efficiency

- Efficiency in terms of compiling and executing a program
- Some concepts related to efficiency
  - Type checking

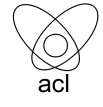


Efficiency

- Efficiency in terms of compiling and executing a program
- Some concepts related to efficiency
  - Type checking
  - Compiler optimization

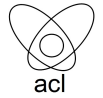


Efficiency

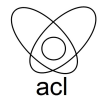
- Efficiency in terms of compiling and executing a program
- Some concepts related to efficiency
  - Type checking
  - Compiler optimization
  - Handling recursions at compile time



Efficiency

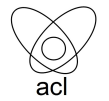
- Efficiency in terms of compiling and executing a program
- Some concepts related to efficiency
  - Type checking
  - Compiler optimization
  - Handling recursions at compile time
  - Interpreted languages vs. compiled languages



Efficiency

- Efficiency in terms of compiling and executing a program
- Some concepts related to efficiency
  - Type checking
  - Compiler optimization
  - Handling recursions at compile time
  - Interpreted languages vs. compiled languages
  - Levels of abstraction



END OF SESSION 3