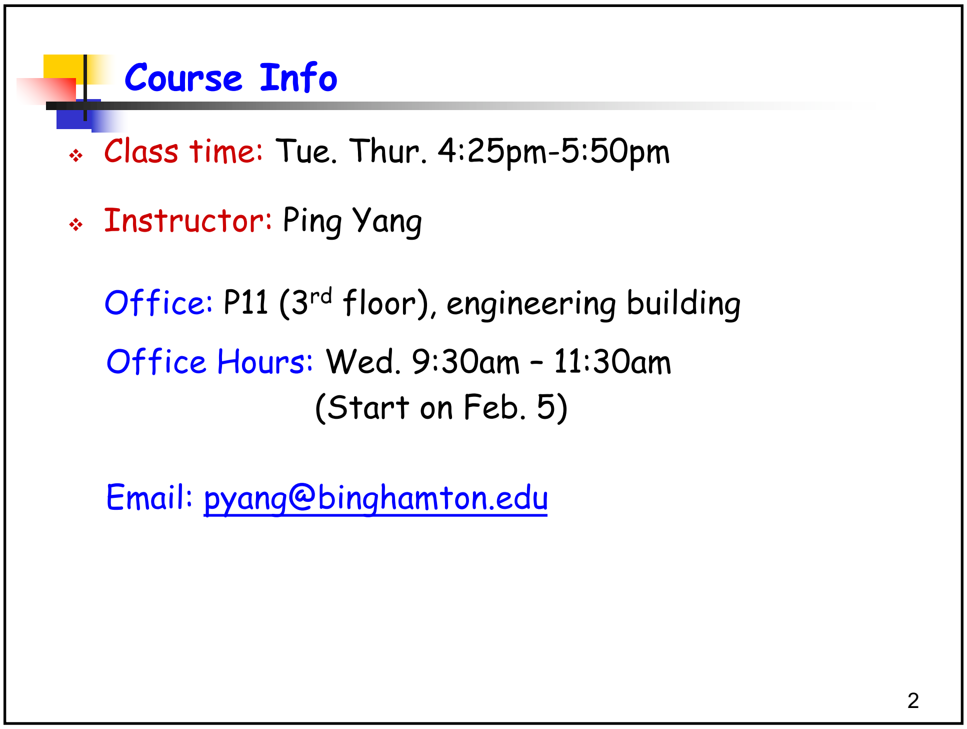
CS571: Programming Languages



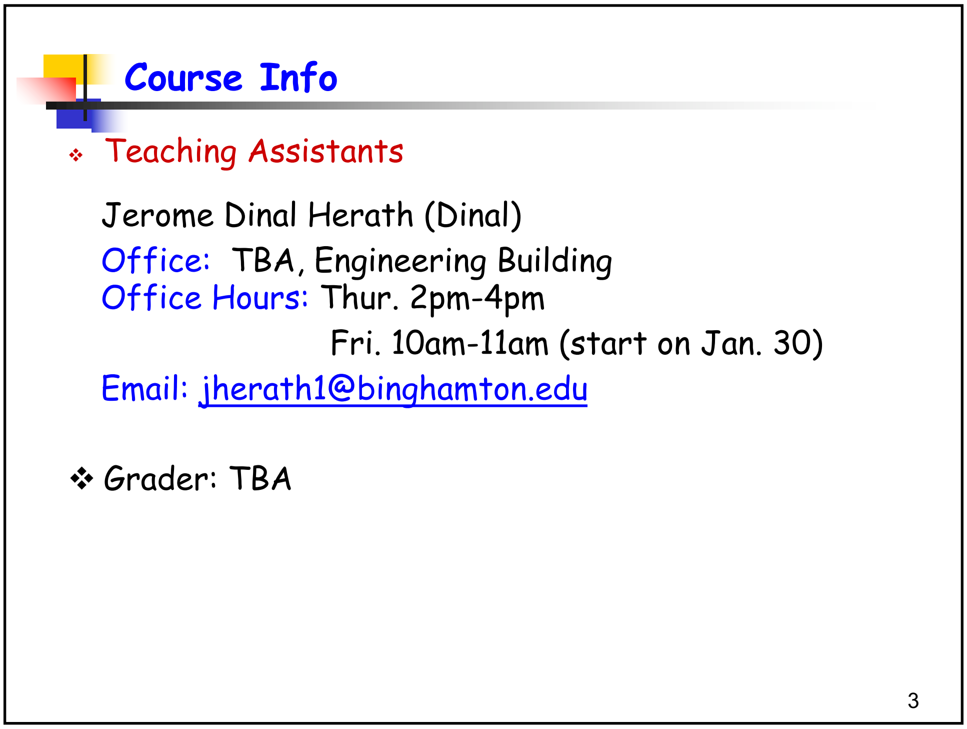
Course Info

- Class time: Tue. Thur. 4:25pm-5:50pm
- Instructor: Ping Yang

  Office: P11 (3rd floor), engineering building

  Office Hours: Wed. 9:30am – 11:30am
  (Start on Feb. 5)

  Email: pyang@binghamton.edu

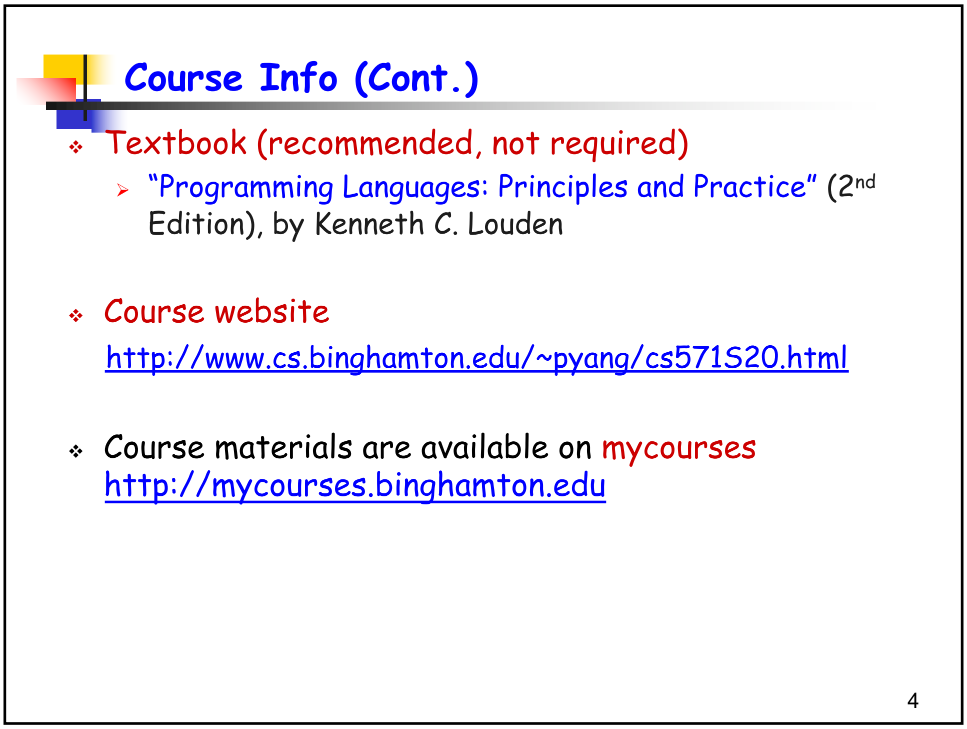


Course Info

- Teaching Assistants

  Jerome Dinal Herath (Dinal)
  Office: TBA, Engineering Building
  Office Hours: Thur. 2pm-4pm
  Fri. 10am-11am (start on Jan. 30)
  Email: jherath1@binghamton.edu

- Grader: TBA



Course Info (Cont.)

- Textbook (recommended, not required)
  - "Programming Languages: Principles and Practice" (2nd Edition), by Kenneth C. Louden

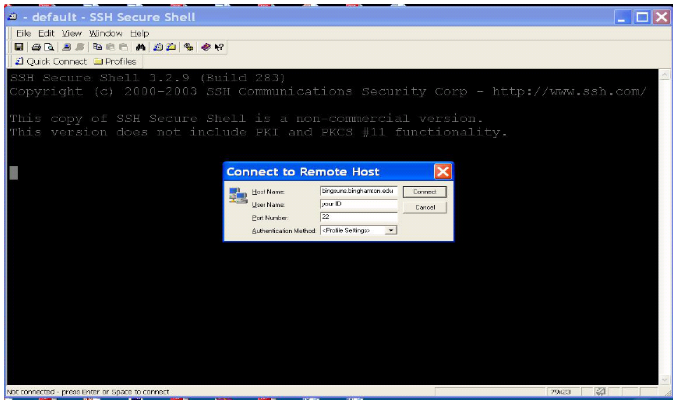
- Course website

  http://www.cs.binghamton.edu/~pyang/cs571S20.html

- Course materials are available on mycourses http://mycourses.binghamton.edu

Course Info (Cont.)

- Make sure that you have an account on bingsuns.binghamton.edu or remote.cs.binghamton.edu
  - Windows: Download SSH secure shell client to access bingsuns
    https://cgi.math.princeton.edu/compudocwiki/index.php?title=HowTos:Connect_to_login_servers_via_ssh



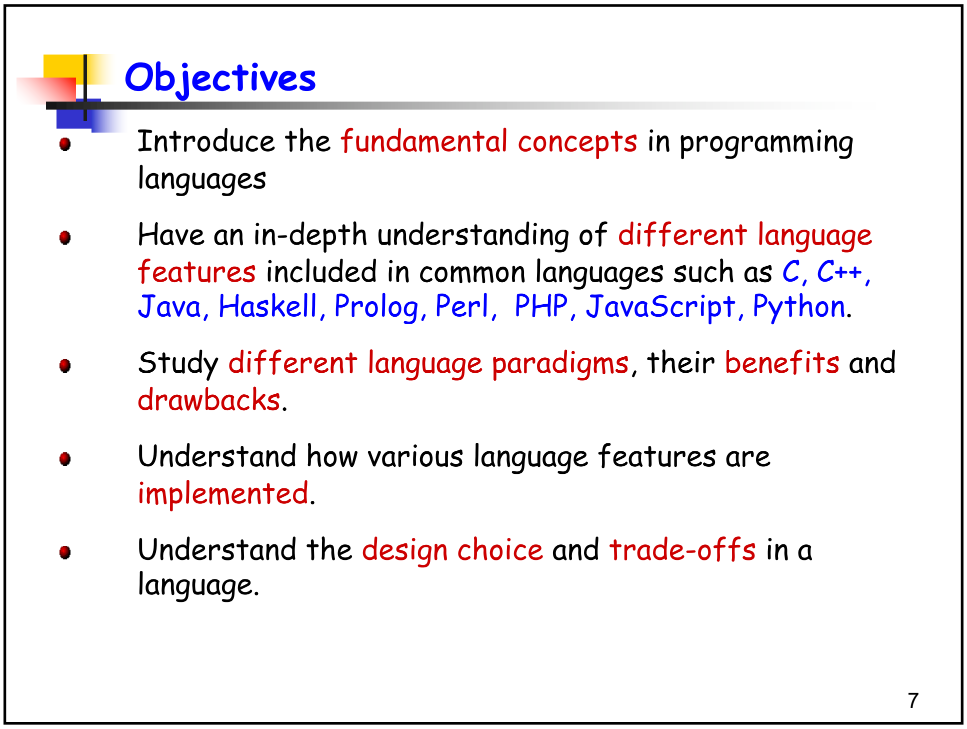
```
SSH Secure Shell 3.2.9 (Build 283)
Copyright (c) 2000-2003 SSH Communications Security Corp - http://www.ssh.com/

This copy of SSH Secure Shell is a non-commercial version.
This version does not include PKI and PKCS #11 functionality.
```

Host Name: bingsuns.binghamton.edu
User Name: your ID
Port Number: 22
Authentication Method: <Profile Settings>

Not connected - press Enter or Space to connect
79x23

Prerequisites

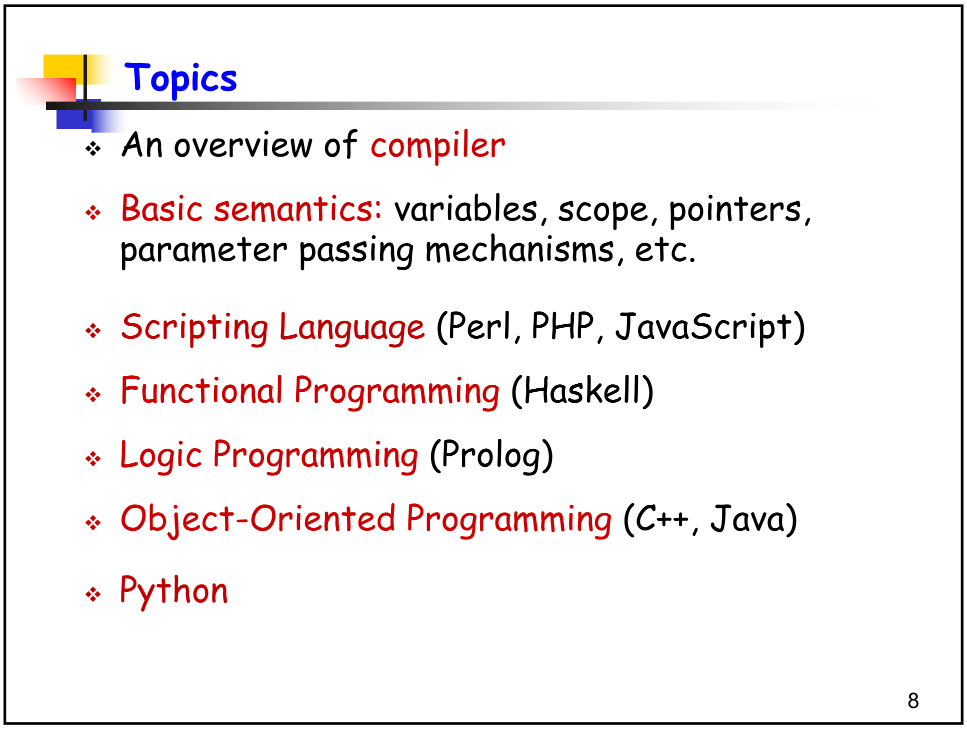
- Proficient with programming in C or C++.
- Comfortable with writing recursive programs.
- Comfortable working and programming in the Unix environment.



Objectives

- Introduce the fundamental concepts in programming languages
- Have an in-depth understanding of different language features included in common languages such as C, C++, Java, Haskell, Prolog, Perl, PHP, JavaScript, Python.
- Study different language paradigms, their benefits and drawbacks.
- Understand how various language features are implemented.
- Understand the design choice and trade-offs in a language.

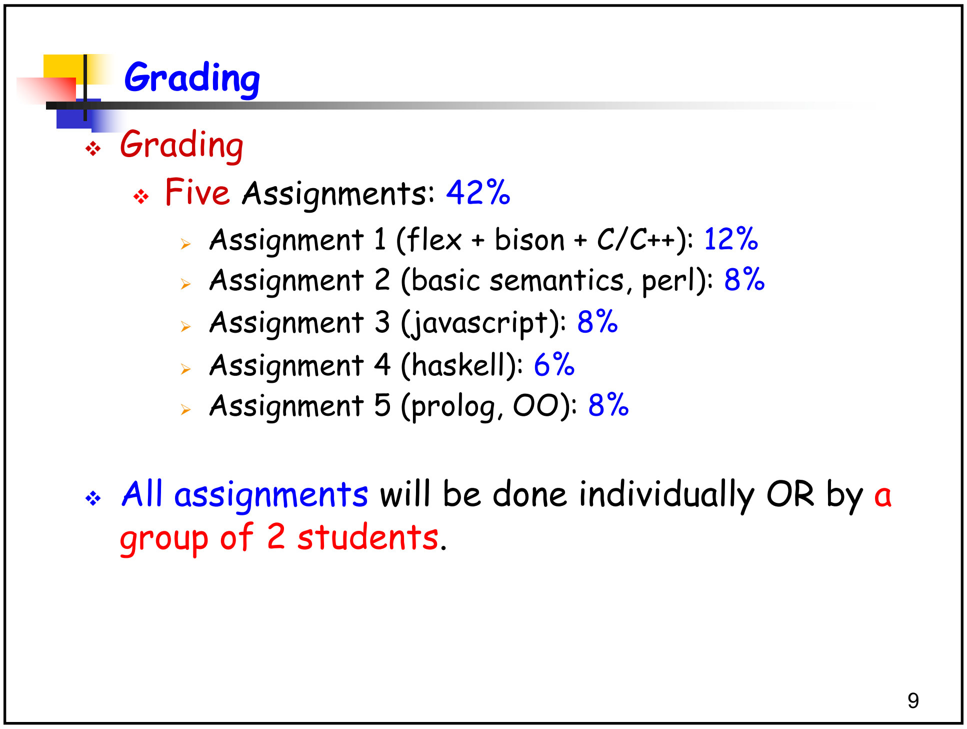
7



Topics

- An overview of compiler
- Basic semantics: variables, scope, pointers, parameter passing mechanisms, etc.
- Scripting Language (Perl, PHP, JavaScript)
- Functional Programming (Haskell)
- Logic Programming (Prolog)
- Object-Oriented Programming (C++, Java)
- Python

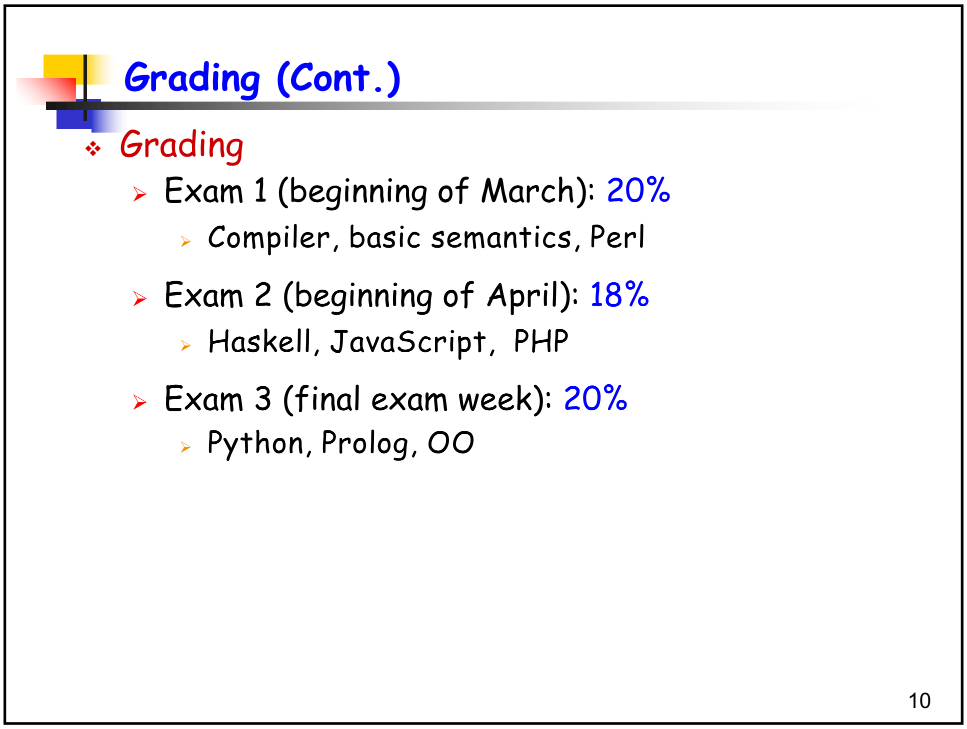
8



Grading

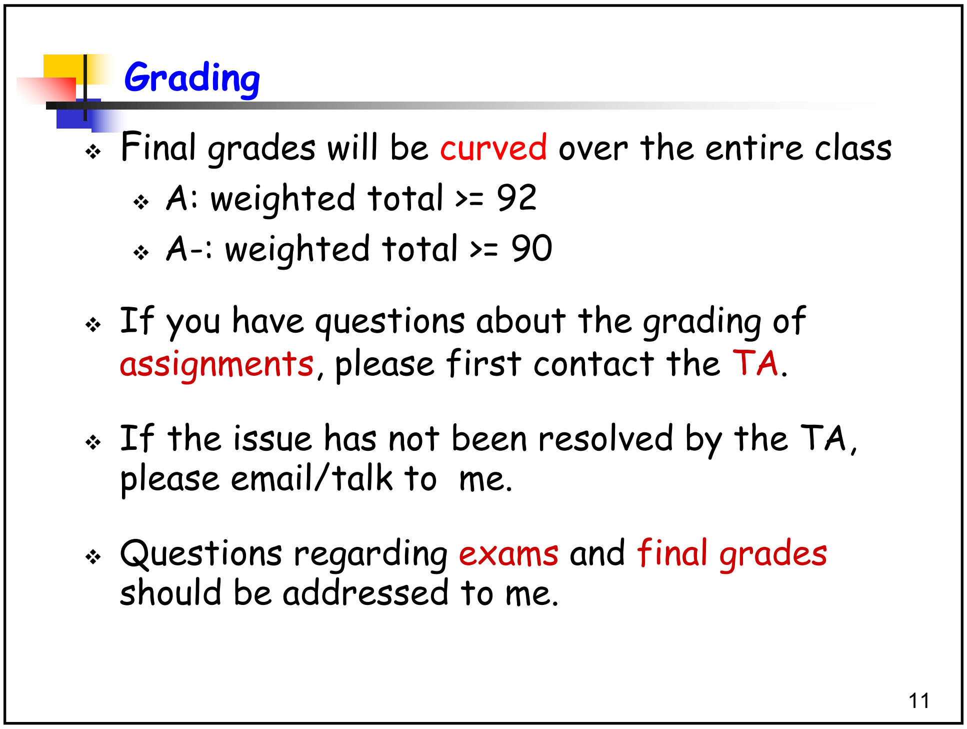
- Grading
  - Five Assignments: 42%
    - Assignment 1 (flex + bison + C/C++): 12%
    - Assignment 2 (basic semantics, perl): 8%
    - Assignment 3 (javascript): 8%
    - Assignment 4 (haskell): 6%
    - Assignment 5 (prolog, OO): 8%

- All assignments will be done individually OR by a group of 2 students.



Grading (Cont.)

- Grading
  - Exam 1 (beginning of March): 20%
    - Compiler, basic semantics, Perl
  - Exam 2 (beginning of April): 18%
    - Haskell, JavaScript, PHP
  - Exam 3 (final exam week): 20%
    - Python, Prolog, OO

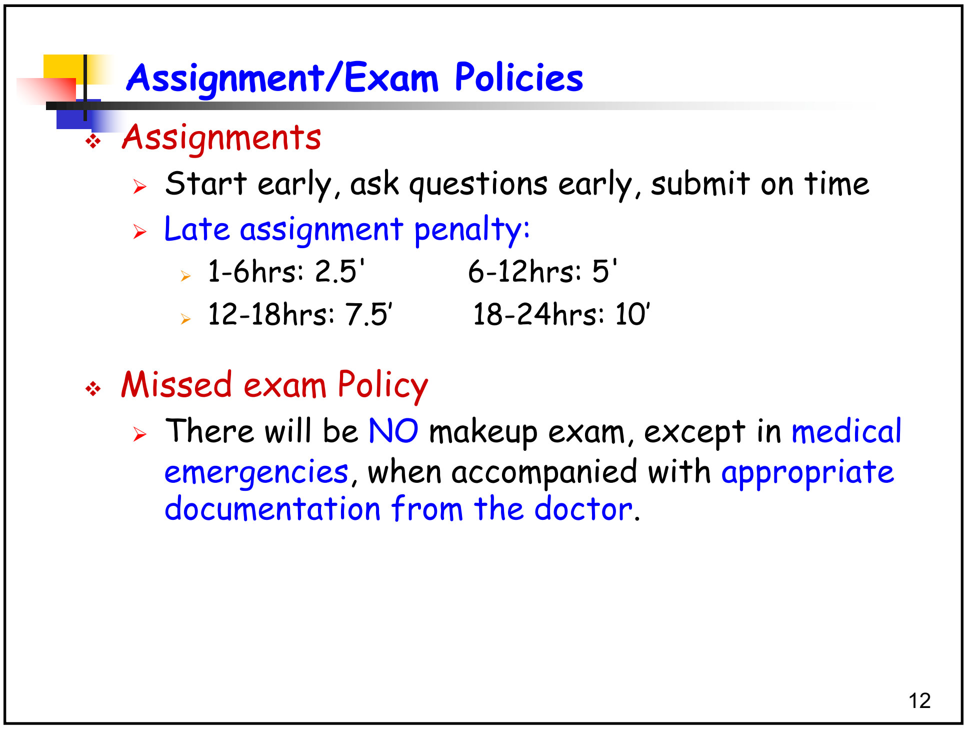


Grading

- Final grades will be curved over the entire class
  - A: weighted total >= 92
  - A-: weighted total >= 90
- If you have questions about the grading of assignments, please first contact the TA.
- If the issue has not been resolved by the TA, please email/talk to me.
- Questions regarding exams and final grades should be addressed to me.

11

11

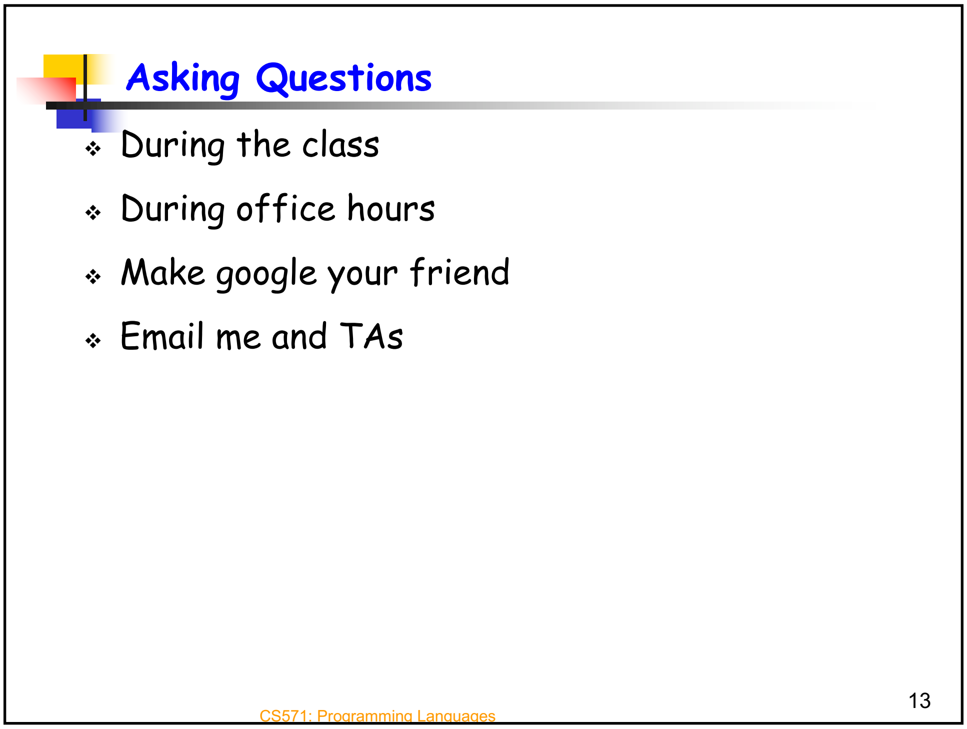


Assignment/Exam Policies

- Assignments
  - Start early, ask questions early, submit on time
  - Late assignment penalty:
    - 1-6hrs: 2.5'   6-12hrs: 5'
    - 12-18hrs: 7.5'   18-24hrs: 10'
- Missed exam Policy
  - There will be NO makeup exam, except in medical emergencies, when accompanied with appropriate documentation from the doctor.

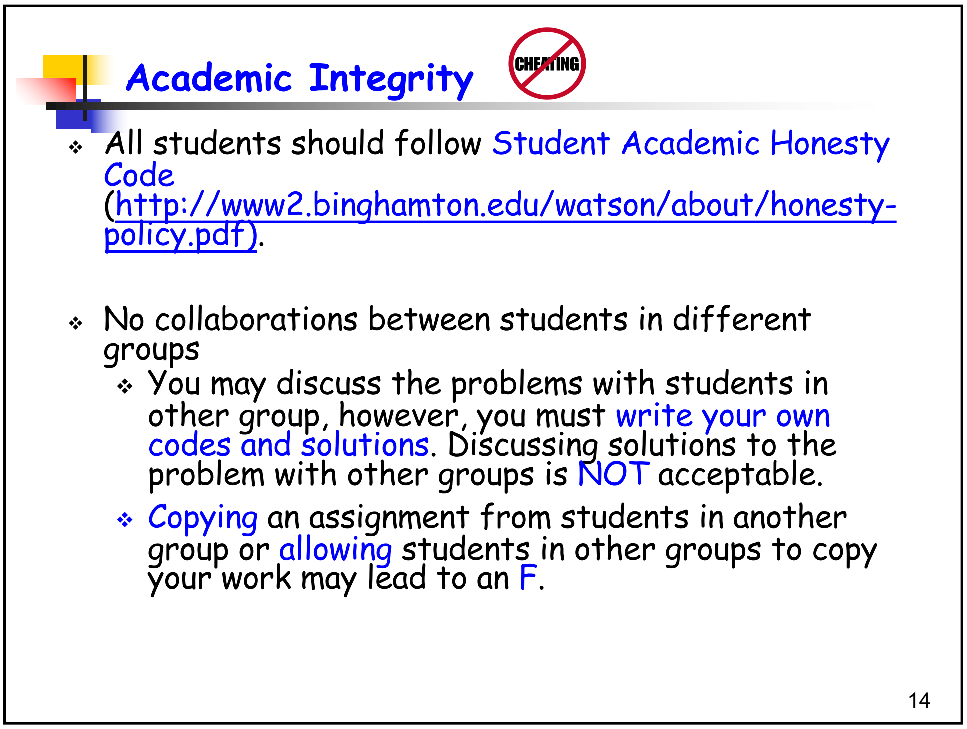
12

12



Asking Questions

- During the class
- During office hours
- Make google your friend
- Email me and TAs

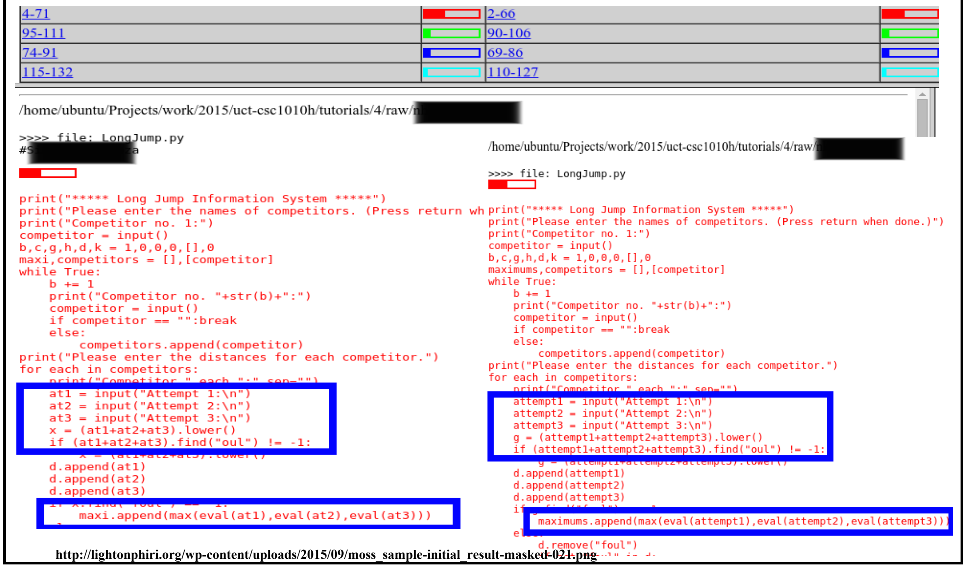


Academic Integrity

- All students should follow Student Academic Honesty Code (http://www2.binghamton.edu/watson/about/honesty-policy.pdf).
- No collaborations between students in different groups
  - You may discuss the problems with students in other group, however, you must write your own codes and solutions. Discussing solutions to the problem with other groups is NOT acceptable.
  - Copying an assignment from students in another group or allowing students in other groups to copy your work may lead to an F.

Academic Integrity

- We will use Moss, to detect plagiarism.



4-71	🟥	2-66	🟥
95-111	🟩	90-106	🟩
74-91	🟦	69-86	🟦
115-132	▭	110-127	▭

/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/4/raw/[redacted]

```
>>>> file: LongJump.py
#[redacted]

print("***** Long Jump Information System *****")
print("Please enter the names of competitors. (Press return wh
print("Competitor no. 1:")
competitor = input()
b,c,g,h,d,k = 1,0,0,0,[],0
maxi,competitors = [],[competitor]
while True:
    b += 1
    print("Competitor no. "+str(b)+":")
    competitor = input()
    if competitor == "":break
    else:
        competitors.append(competitor)
print("Please enter the distances for each competitor.")
for each in competitors:
    print("Competitor ", each ":" sep="")
    at1 = input("Attempt 1:\n")
    at2 = input("Attempt 2:\n")
    at3 = input("Attempt 3:\n")
    x = (at1+at2+at3).lower()
    if (at1+at2+at3).find("oul") != -1:
        x = (at1+at2+at3).lower()
    d.append(at1)
    d.append(at2)
    d.append(at3)
    if x.find("foul") == -1:
        maxi.append(max(eval(at1),eval(at2),eval(at3)))
```

/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/4/raw/[redacted]

```
>>>> file: LongJump.py

print("***** Long Jump Information System *****")
print("Please enter the names of competitors. (Press return when done.)")
print("Competitor no. 1:")
competitor = input()
b,c,g,h,d,k = 1,0,0,0,[],0
maximums,competitors = [],[competitor]
while True:
    b += 1
    print("Competitor no. "+str(b)+":")
    competitor = input()
    if competitor == "":break
    else:
        competitors.append(competitor)
print("Please enter the distances for each competitor.")
for each in competitors:
    print("Competitor ", each ":" sep="")
    attempt1 = input("Attempt 1:\n")
    attempt2 = input("Attempt 2:\n")
    attempt3 = input("Attempt 3:\n")
    g = (attempt1+attempt2+attempt3).lower()
    if (attempt1+attempt2+attempt3).find("oul") != -1:
        g = (attempt1+attempt2+attempt3).lower()
    d.append(attempt1)
    d.append(attempt2)
    d.append(attempt3)
    if
        maximums.append(max(eval(attempt1),eval(attempt2),eval(attempt3)))
    else:
        d.remove("foul")
```

http://lightonphiri.org/wp-content/uploads/2015/09/moss_sample-initial_result-masked-021.png

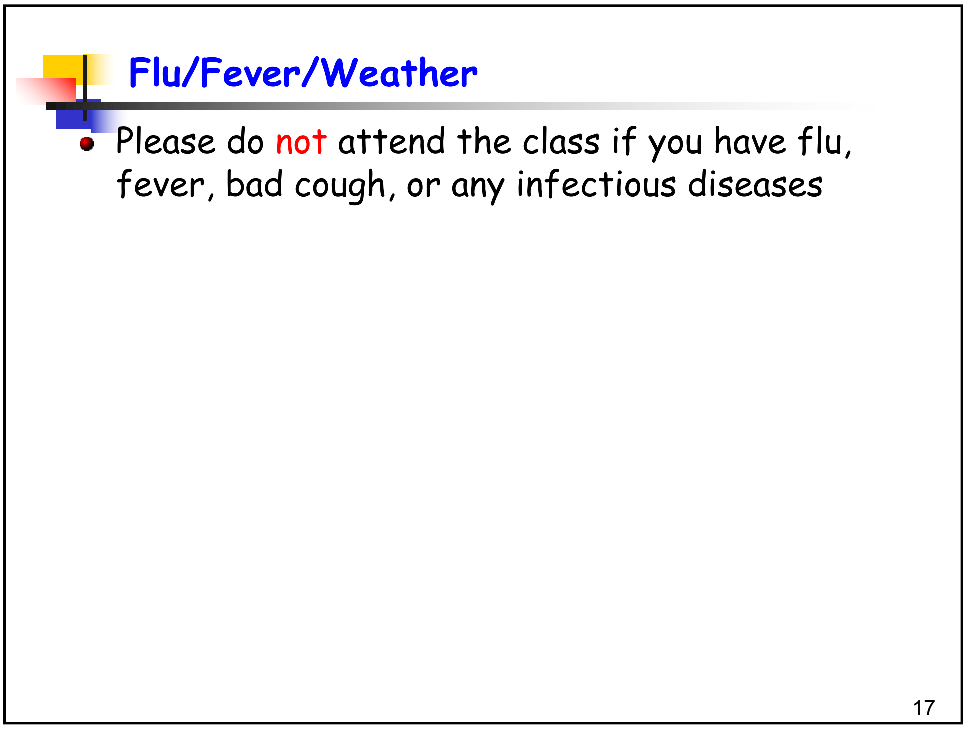
15

Academic Integrity

- Use chmod 700 <directoryname> command to change the permissions of your working directories before you start working on the assignments.
- Copying materials from the Internet will be considered academic dishonesty.

16

16

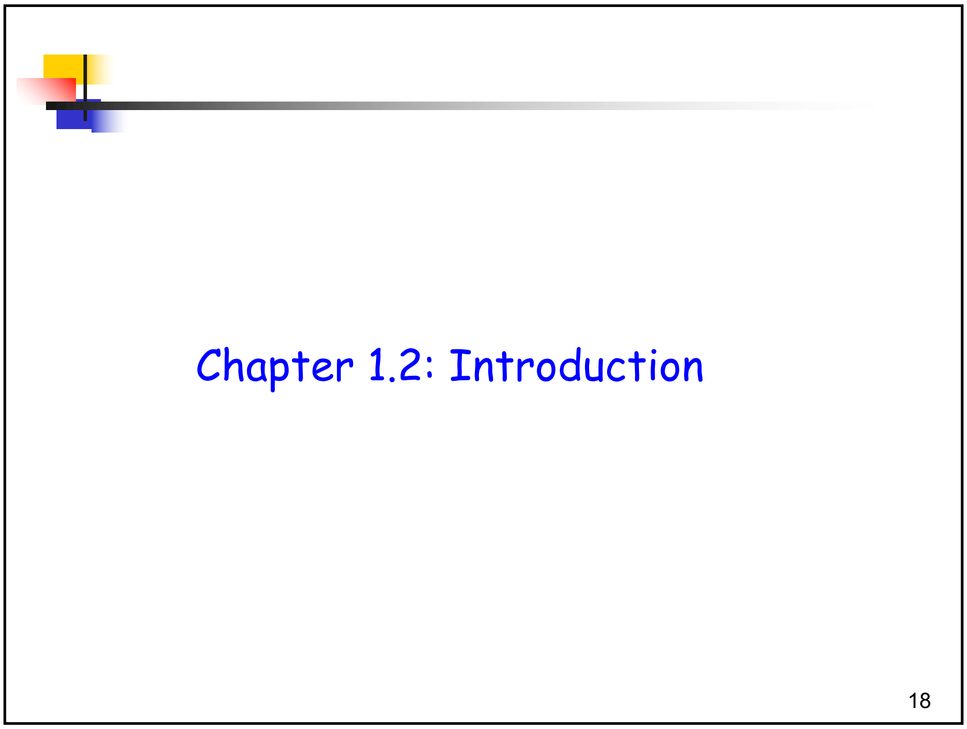


Flu/Fever/Weather

- Please do not attend the class if you have flu, fever, bad cough, or any infectious diseases

17

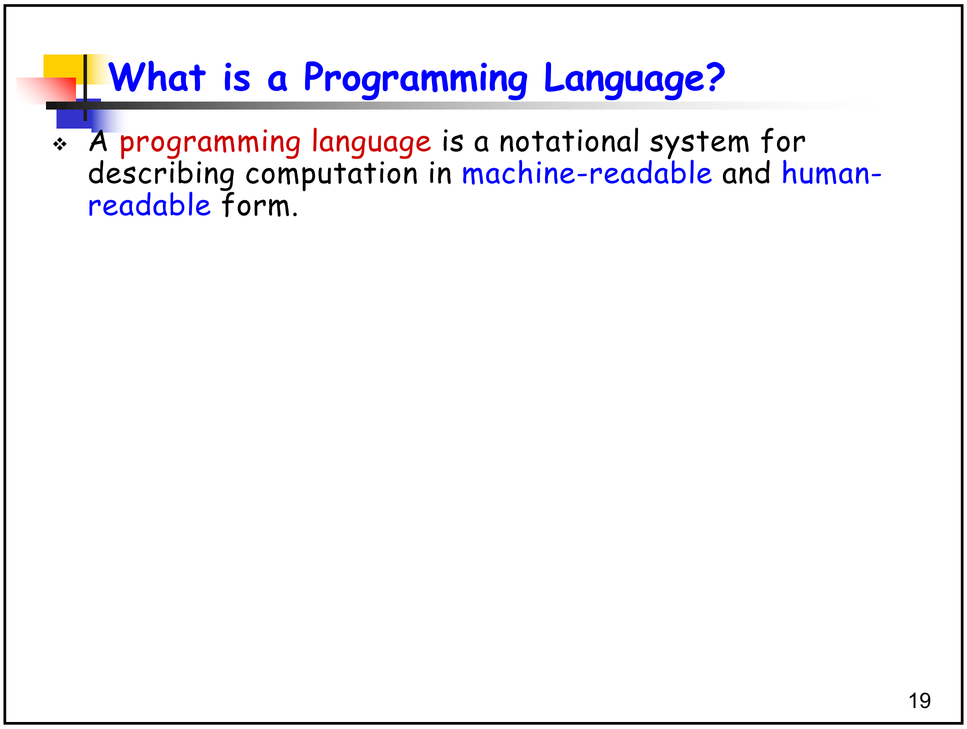
17



Chapter 1.2: Introduction

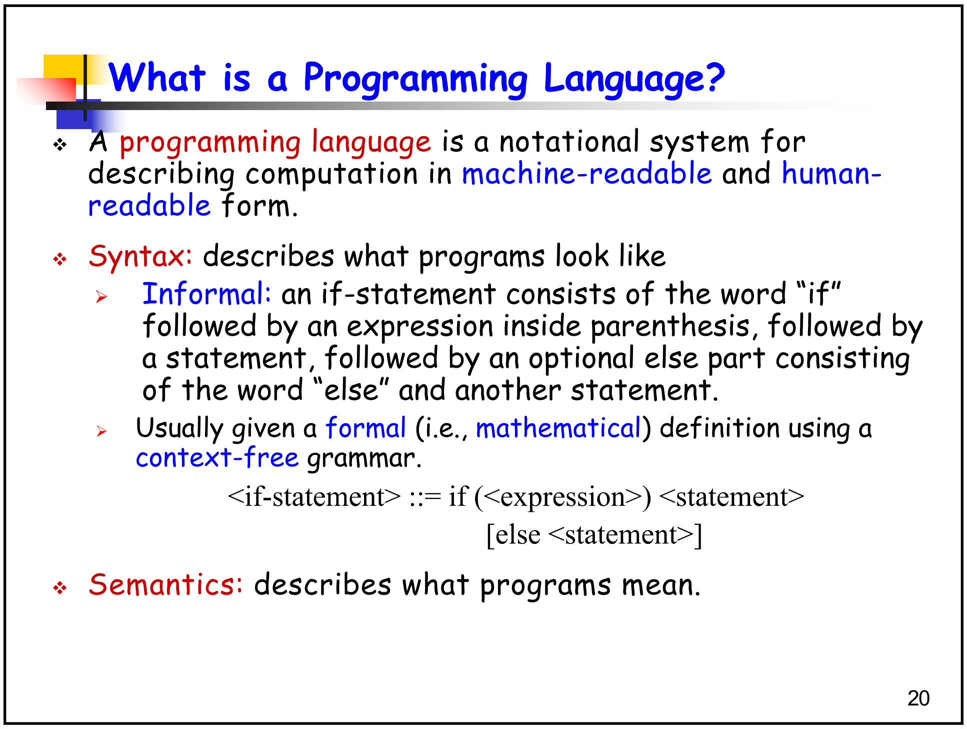
18

18



What is a Programming Language?

- A programming language is a notational system for describing computation in machine-readable and human-readable form.



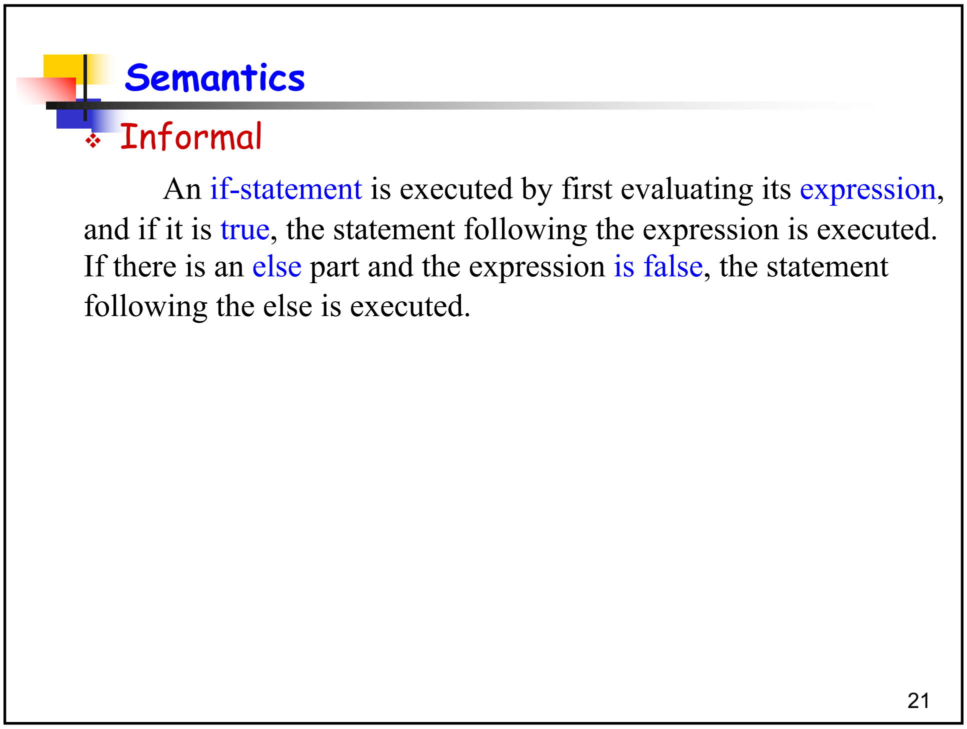
What is a Programming Language?

- A programming language is a notational system for describing computation in machine-readable and human-readable form.

- Syntax: describes what programs look like
  - Informal: an if-statement consists of the word "if" followed by an expression inside parenthesis, followed by a statement, followed by an optional else part consisting of the word "else" and another statement.
  - Usually given a formal (i.e., mathematical) definition using a context-free grammar.

```
<if-statement> ::= if (<expression>) <statement>
                          [else <statement>]
```

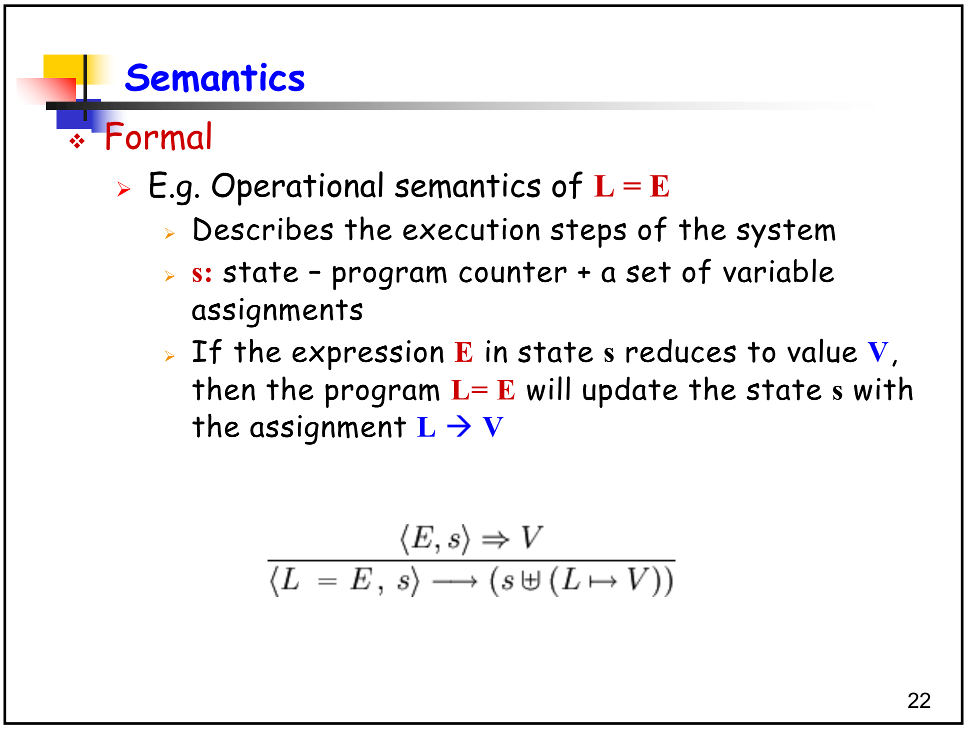
- Semantics: describes what programs mean.



Semantics

- Informal

An if-statement is executed by first evaluating its expression, and if it is true, the statement following the expression is executed. If there is an else part and the expression is false, the statement following the else is executed.



Semantics

- Formal
  - E.g. Operational semantics of **L = E**
    - Describes the execution steps of the system
    - **s:** state – program counter + a set of variable assignments
    - If the expression **E** in state s reduces to value **V**, then the program **L= E** will update the state s with the assignment **L → V**

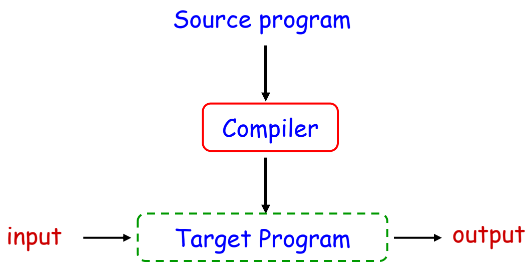
$$\frac{\langle E, s\rangle \Rightarrow V}{\langle L \; = E\,,\; s\rangle \longrightarrow (s \uplus (L \mapsto V))}$$

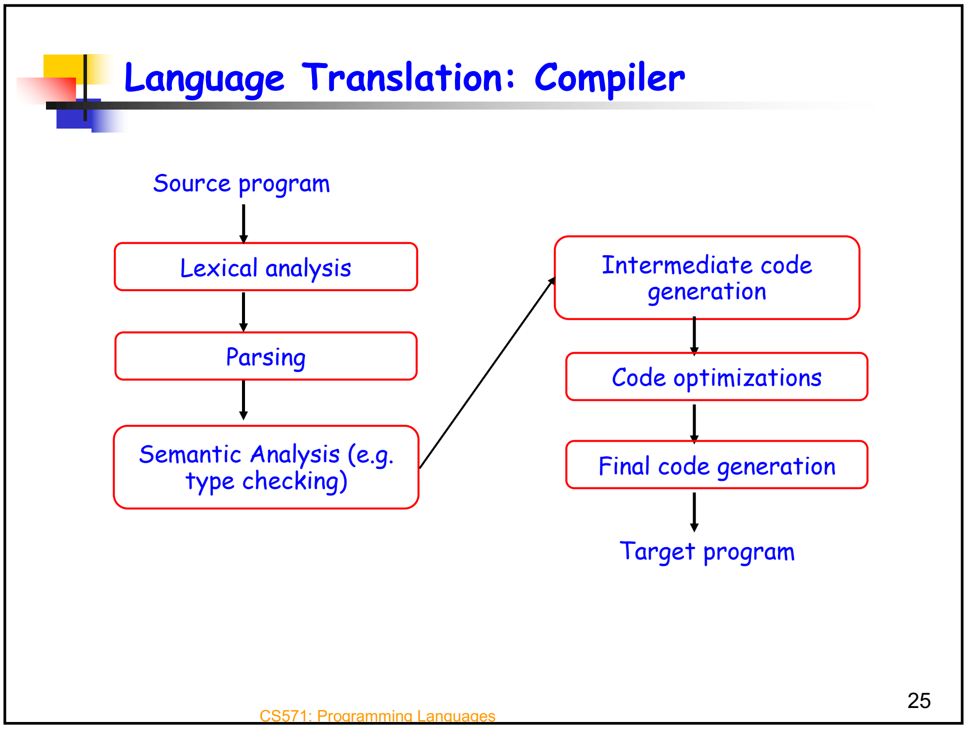
Language Translation

- Programming problems are easier to solve in high-level languages.
- High-level programming languages are not machine-readable -- we need to have a translator.

Language Translation: Compiler

- Compiler
  - translates source code into target code
  - The user may execute the target code.





Language Translation: Compiler

Source program

Lexical analysis

Parsing

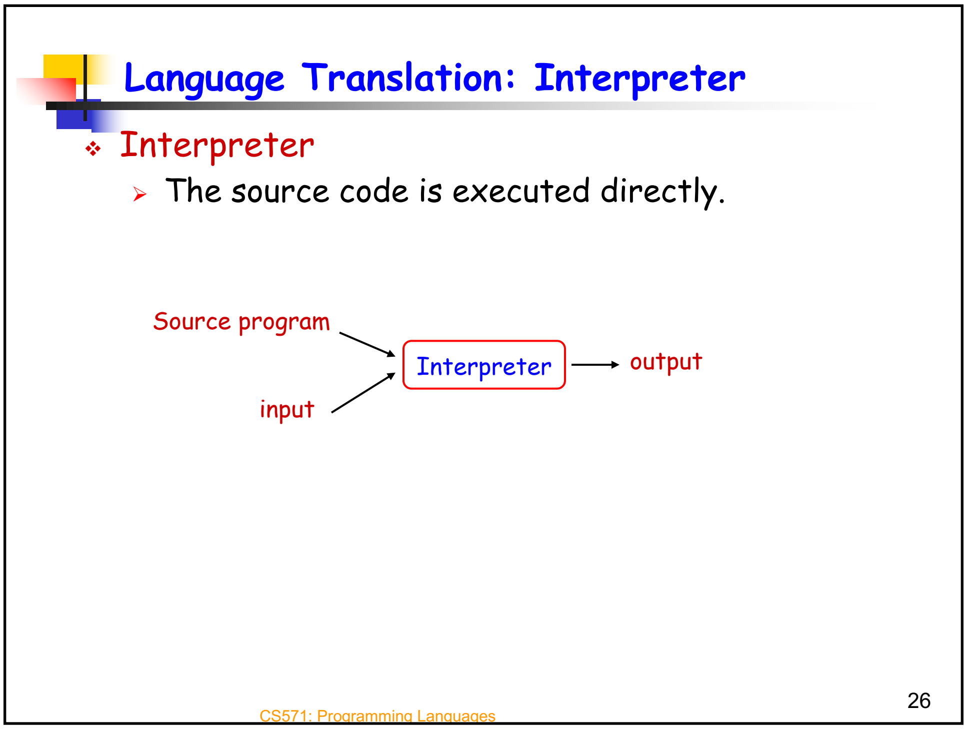
Semantic Analysis (e.g. type checking)

Intermediate code generation

Code optimizations

Final code generation

Target program



Language Translation: Interpreter

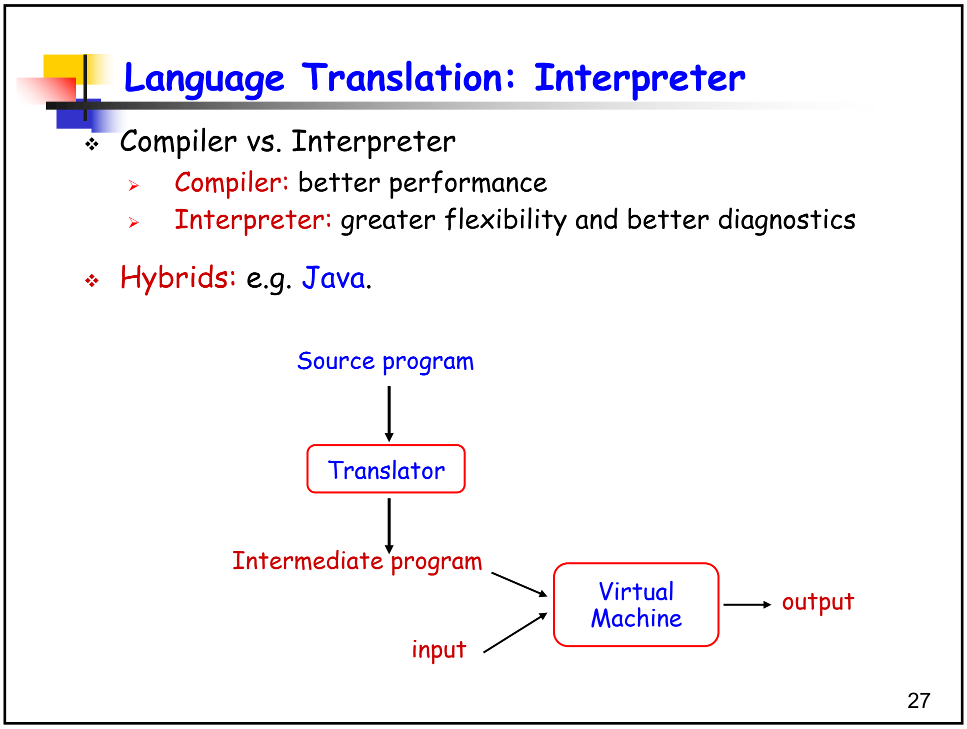
- Interpreter
  - The source code is executed directly.

Source program

input

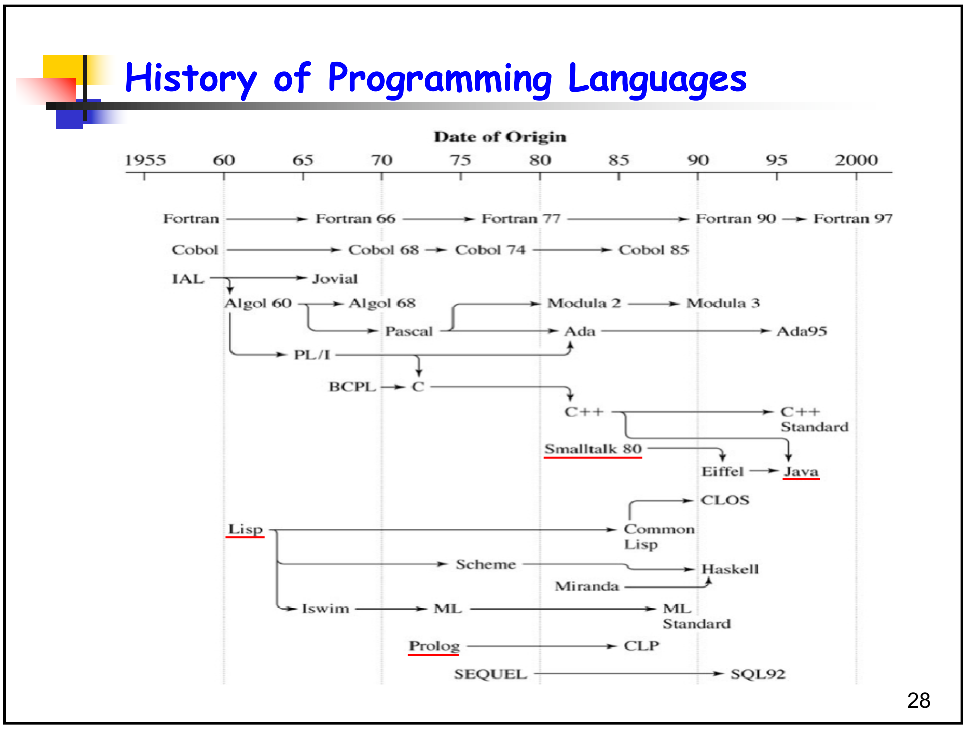
Interpreter

output

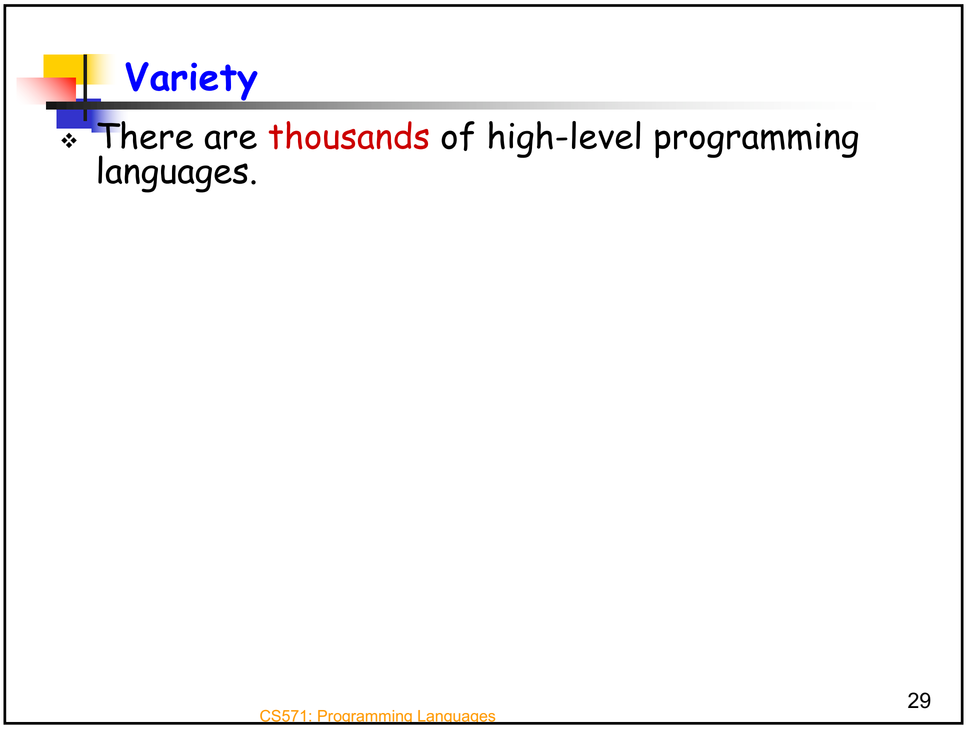


Language Translation: Interpreter

- Compiler vs. Interpreter
  - Compiler: better performance
  - Interpreter: greater flexibility and better diagnostics
- Hybrids: e.g. Java.



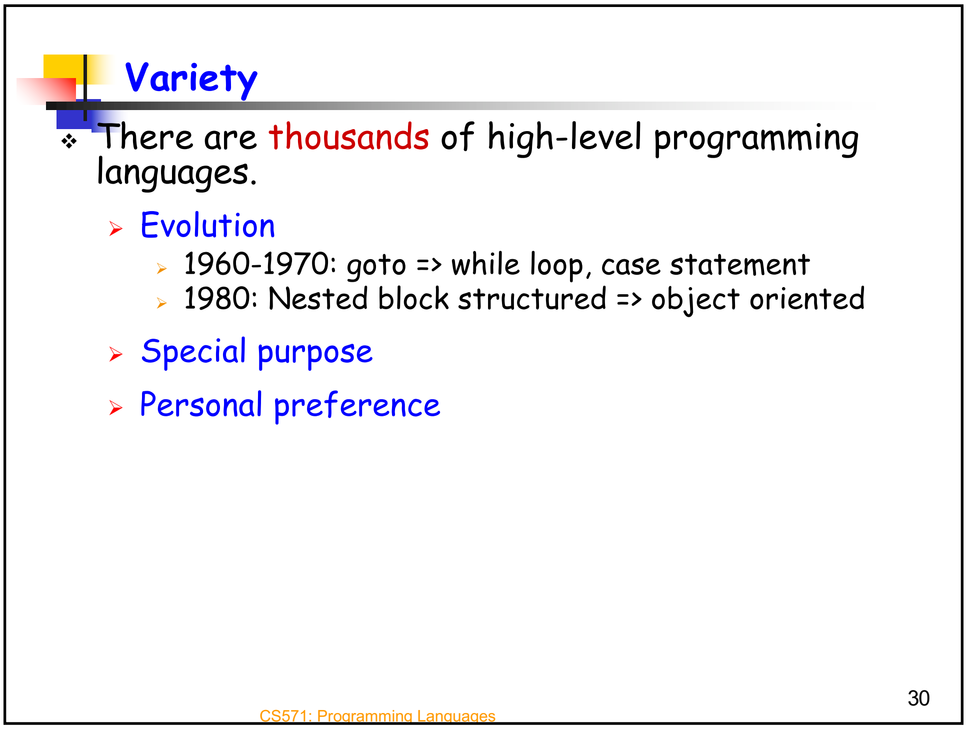
History of Programming Languages



Variety

- There are thousands of high-level programming languages.

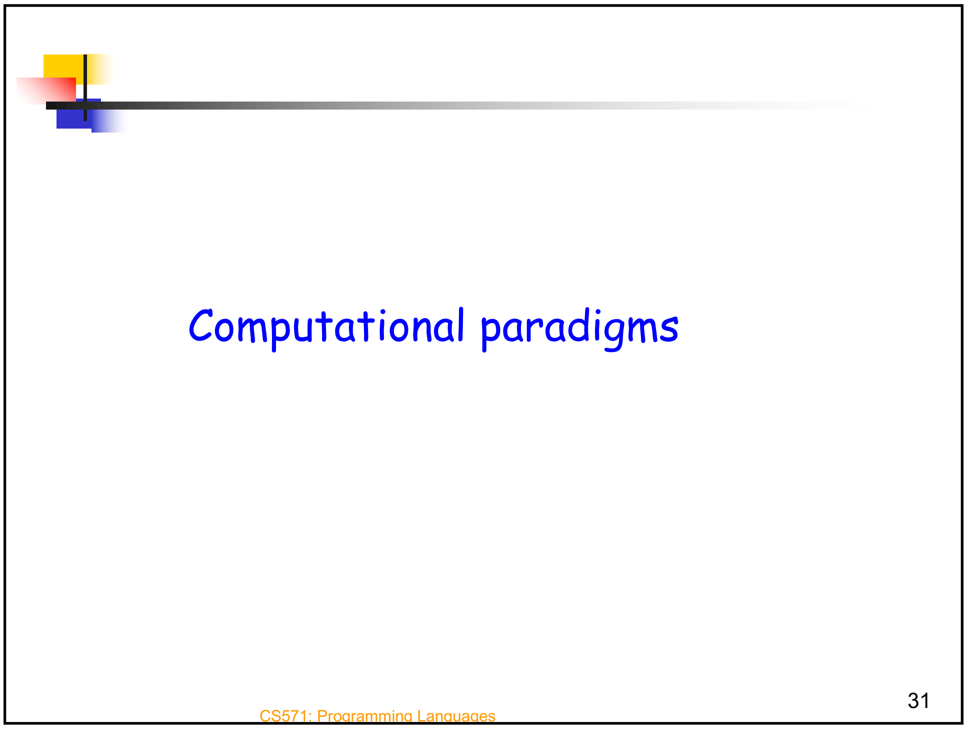
29



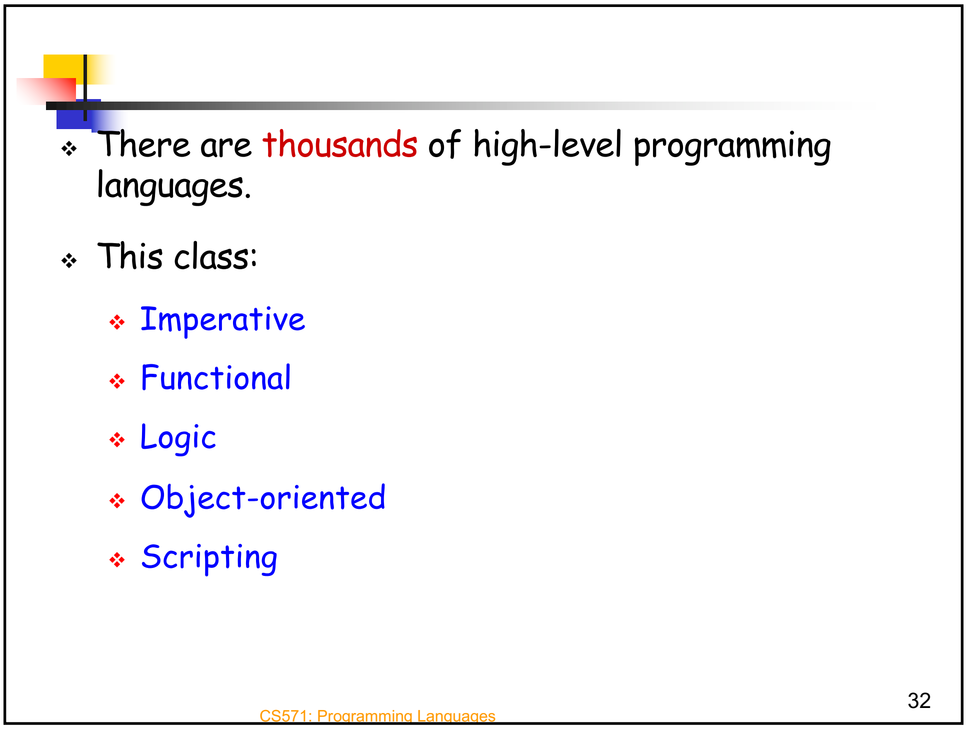
Variety

- There are thousands of high-level programming languages.
  - Evolution
    - 1960-1970: goto => while loop, case statement
    - 1980: Nested block structured => object oriented
  - Special purpose
  - Personal preference

30



Computational paradigms



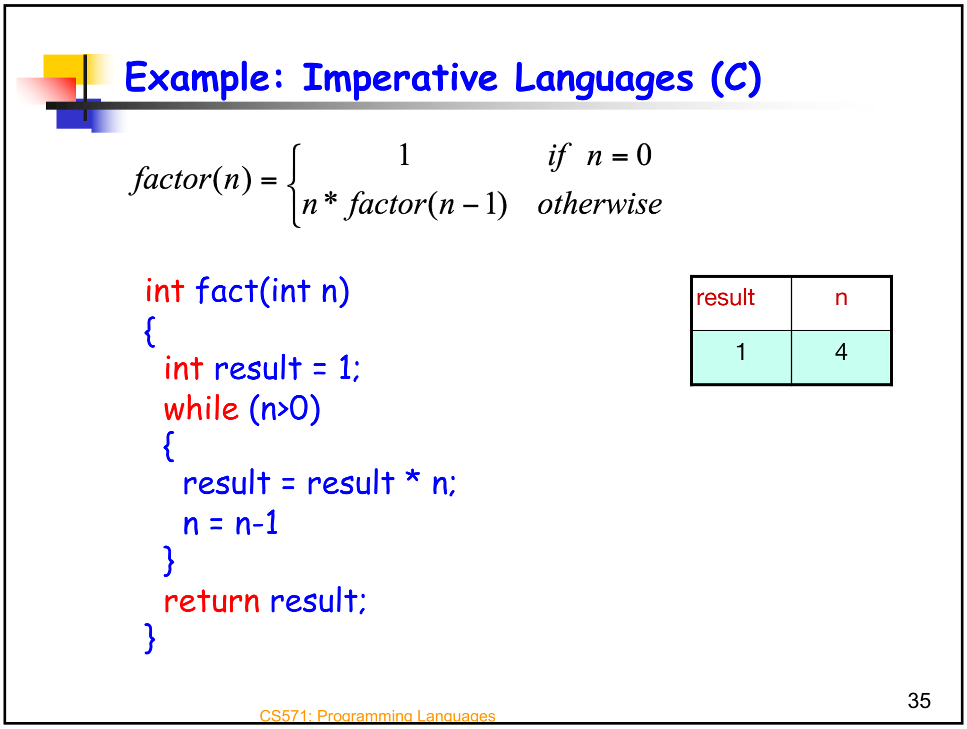
- There are thousands of high-level programming languages.
- This class:
  - Imperative
  - Functional
  - Logic
  - Object-oriented
  - Scripting

Imperative (Procedural) Languages

- Tell a computer what to do at each step.
- The hardware implementation of almost all computers is imperative
- Features:
  - The sequential execution of instructions – order of execution is critical.
  - The use of variables representing memory locations
  - The use of assignment to change the value of variables
- C, Pascal, core Ada, FORTRAN

Example: Imperative Languages (C)

$$factor(n) = \begin{cases} 1 & if \;\; n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

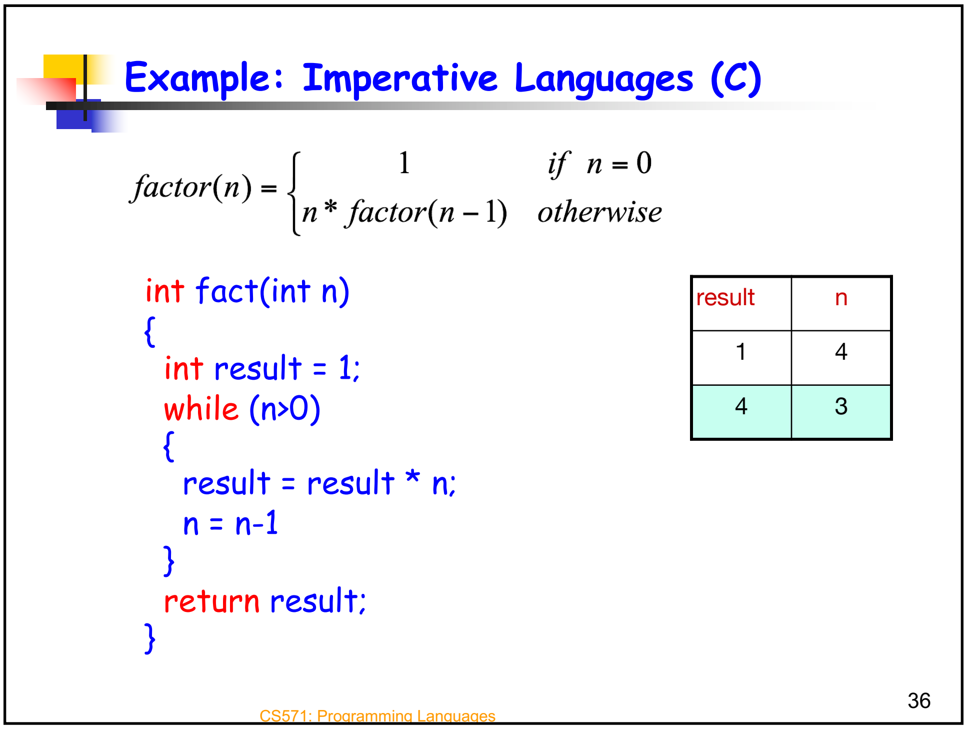


Example: Imperative Languages (C)

$$factor(n) = \begin{cases} 1 & if\ \ n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

```
int fact(int n)
{
  int result = 1;
  while (n>0)
  {
    result = result * n;
    n = n-1
  }
  return result;
}
```

result	n
1	4

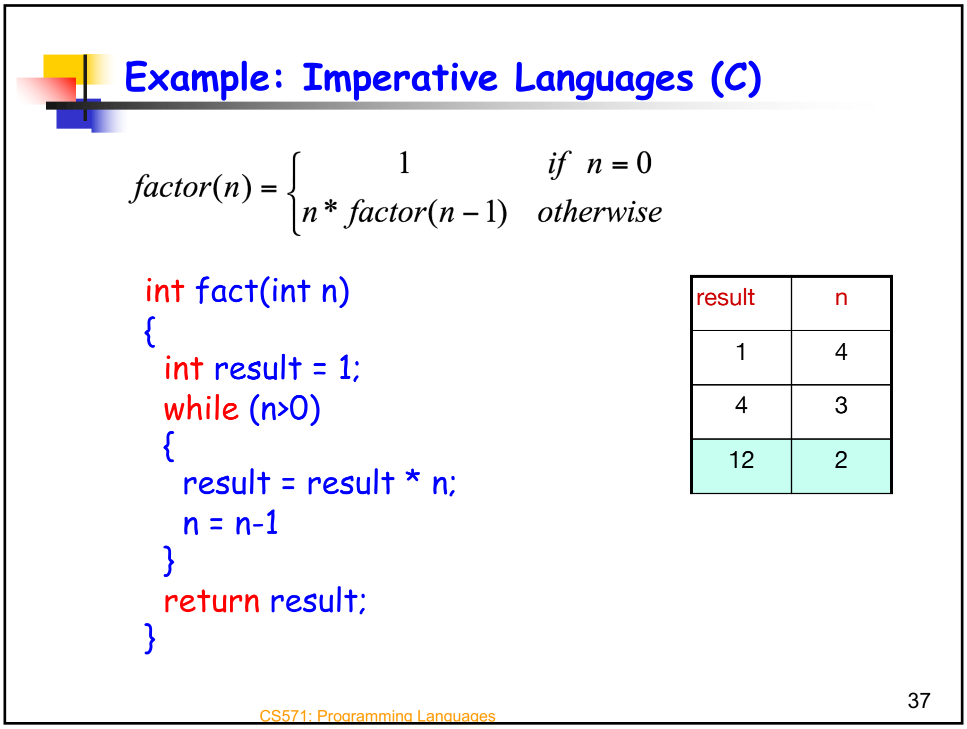


Example: Imperative Languages (C)

$$factor(n) = \begin{cases} 1 & if\ \ n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

```
int fact(int n)
{
  int result = 1;
  while (n>0)
  {
    result = result * n;
    n = n-1
  }
  return result;
}
```

result	n
1	4
4	3



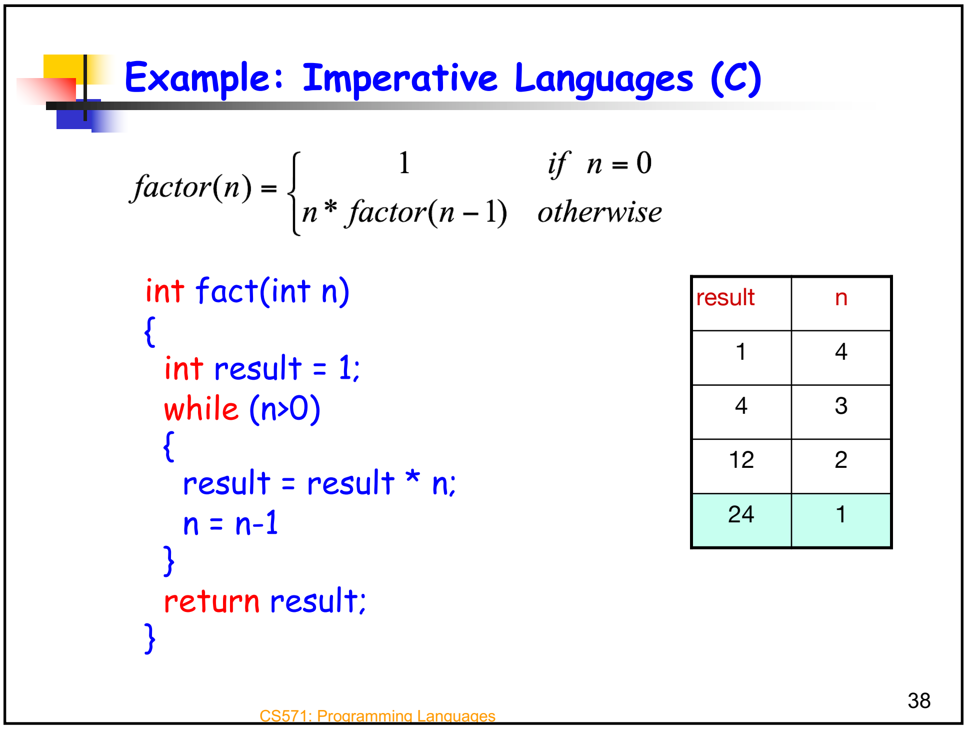
Example: Imperative Languages (C)

$$factor(n) = \begin{cases} 1 & if \;\; n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

```
int fact(int n)
{
  int result = 1;
  while (n>0)
  {
    result = result * n;
    n = n-1
  }
  return result;
}
```

result	n
1	4
4	3
12	2

37



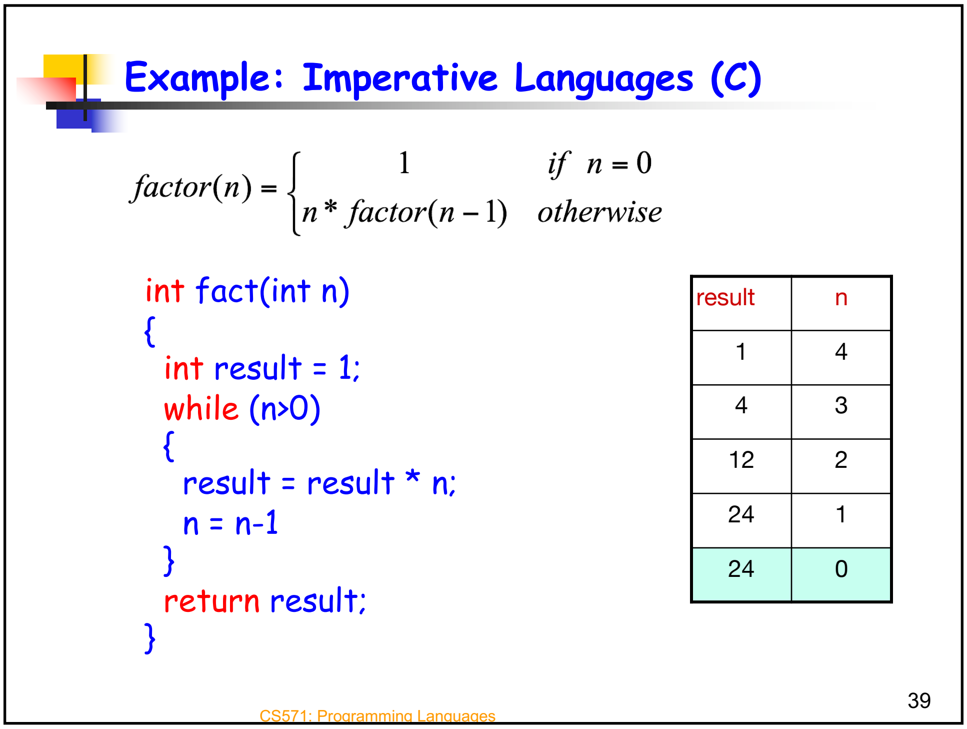
Example: Imperative Languages (C)

$$factor(n) = \begin{cases} 1 & if \;\; n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

```
int fact(int n)
{
  int result = 1;
  while (n>0)
  {
    result = result * n;
    n = n-1
  }
  return result;
}
```

result	n
1	4
4	3
12	2
24	1

38



Example: Imperative Languages (C)

$$factor(n) = \begin{cases} 1 & if \;\; n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

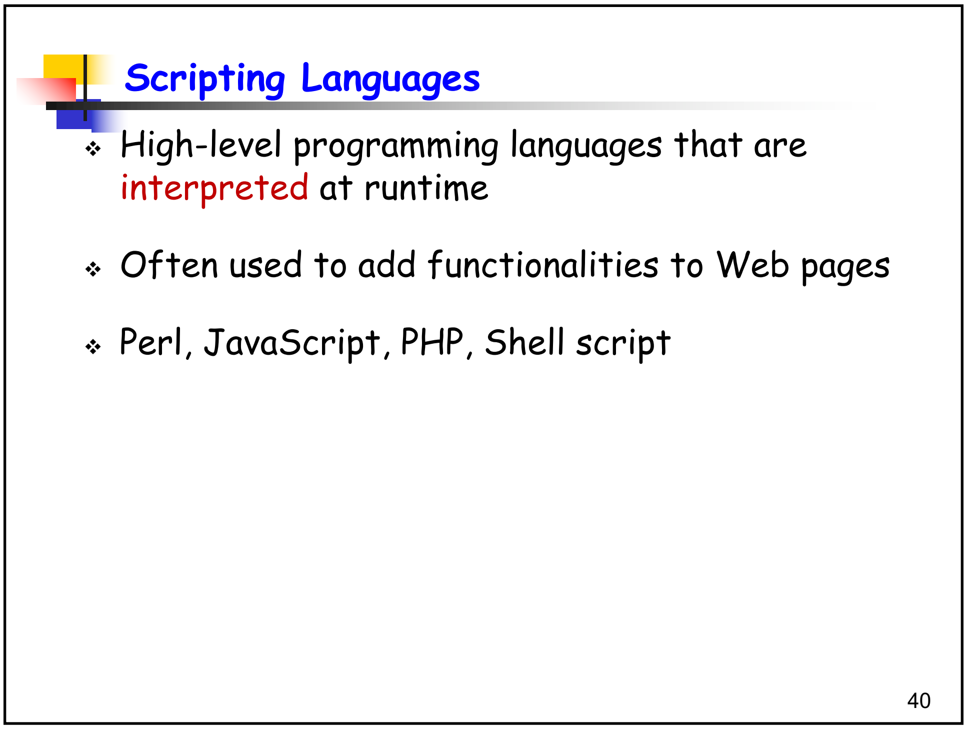
```
int fact(int n)
{
  int result = 1;
  while (n>0)
  {
    result = result * n;
    n = n-1
  }
  return result;
}
```

result	n
1	4
4	3
12	2
24	1
24	0

CS571: Programming Languages

39

39

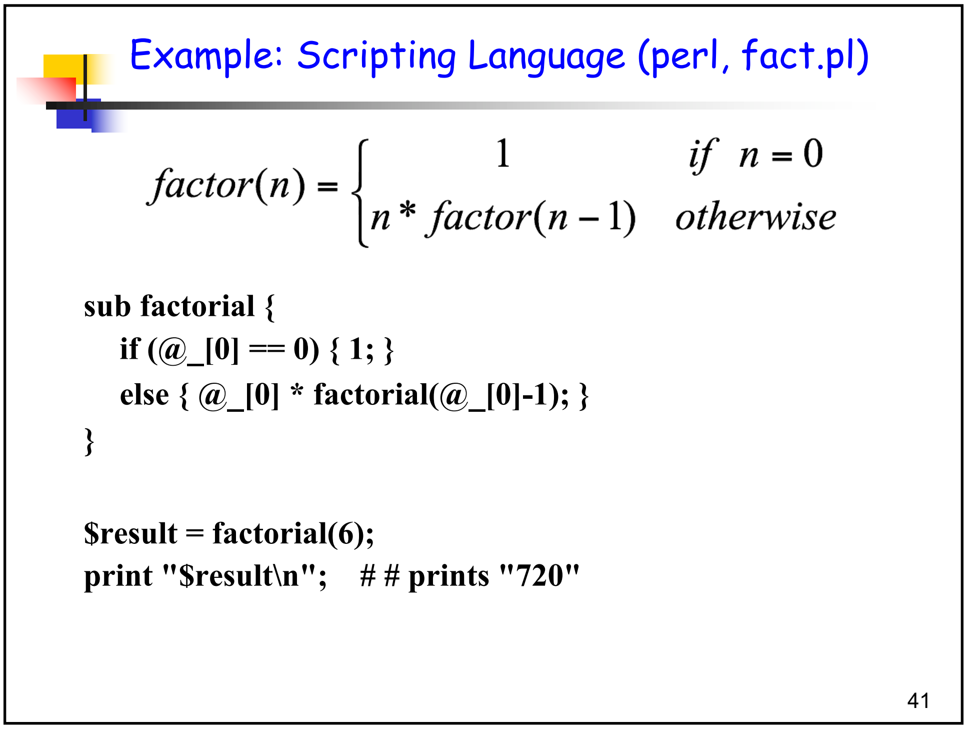


Scripting Languages

- High-level programming languages that are interpreted at runtime
- Often used to add functionalities to Web pages
- Perl, JavaScript, PHP, Shell script

40

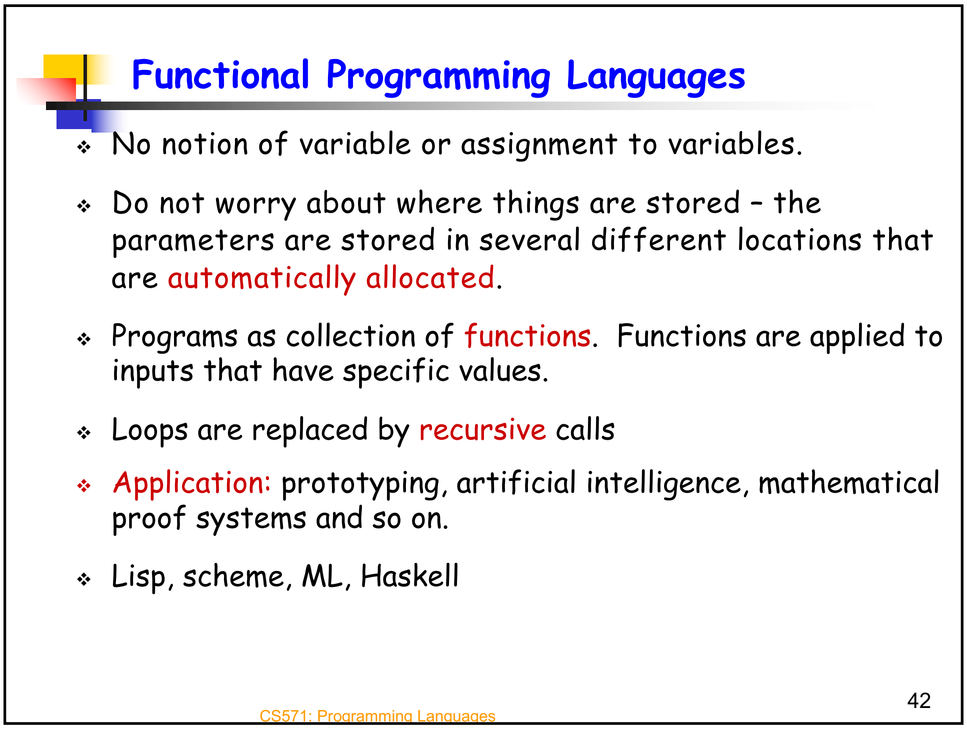
40



Example: Scripting Language (perl, fact.pl)

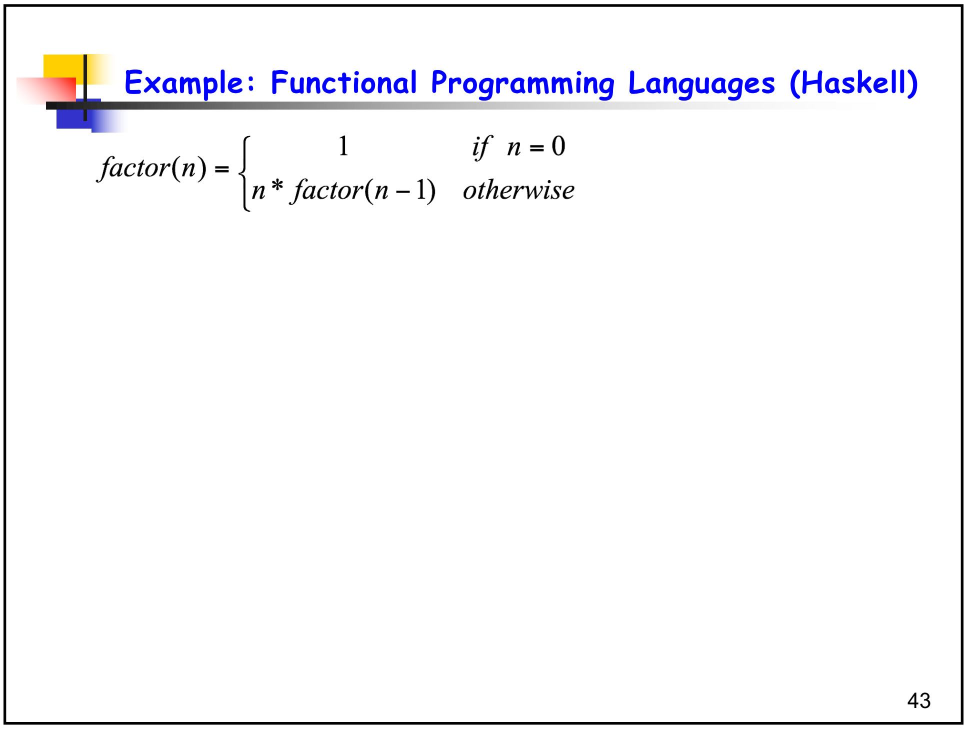
$$factor(n) = \begin{cases} 1 & if \;\; n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

```
sub factorial {
   if (@_[0] == 0) { 1; }
   else { @_[0] * factorial(@_[0]-1); }
}

$result = factorial(6);
print "$result\n";    # # prints "720"
```



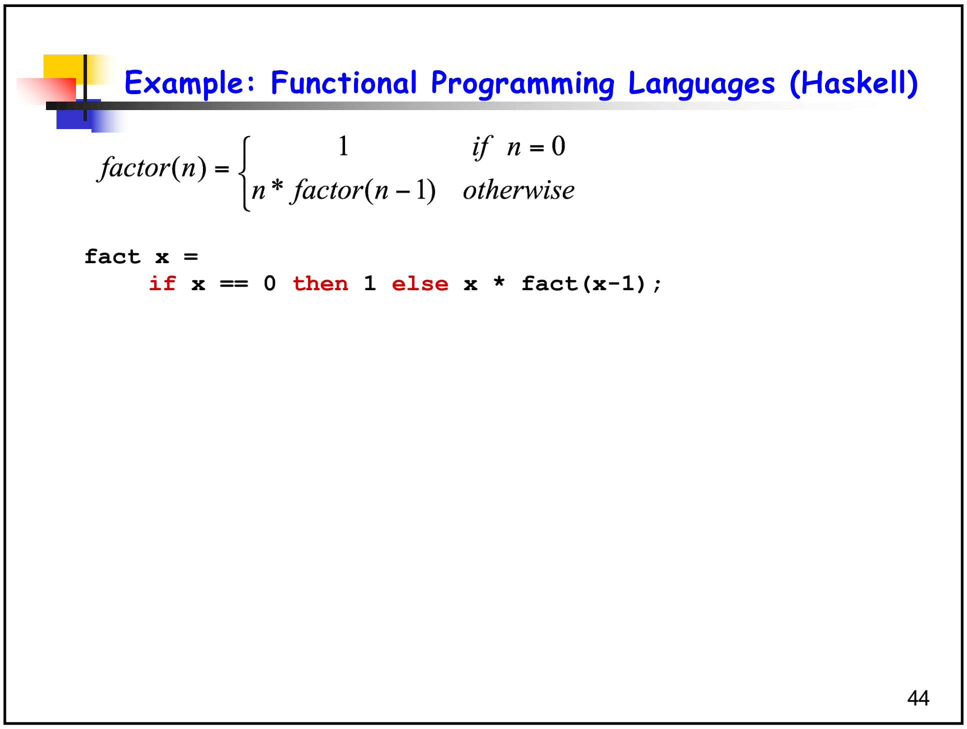
Functional Programming Languages

- No notion of variable or assignment to variables.
- Do not worry about where things are stored – the parameters are stored in several different locations that are automatically allocated.
- Programs as collection of functions. Functions are applied to inputs that have specific values.
- Loops are replaced by recursive calls
- Application: prototyping, artificial intelligence, mathematical proof systems and so on.
- Lisp, scheme, ML, Haskell



Example: Functional Programming Languages (Haskell)

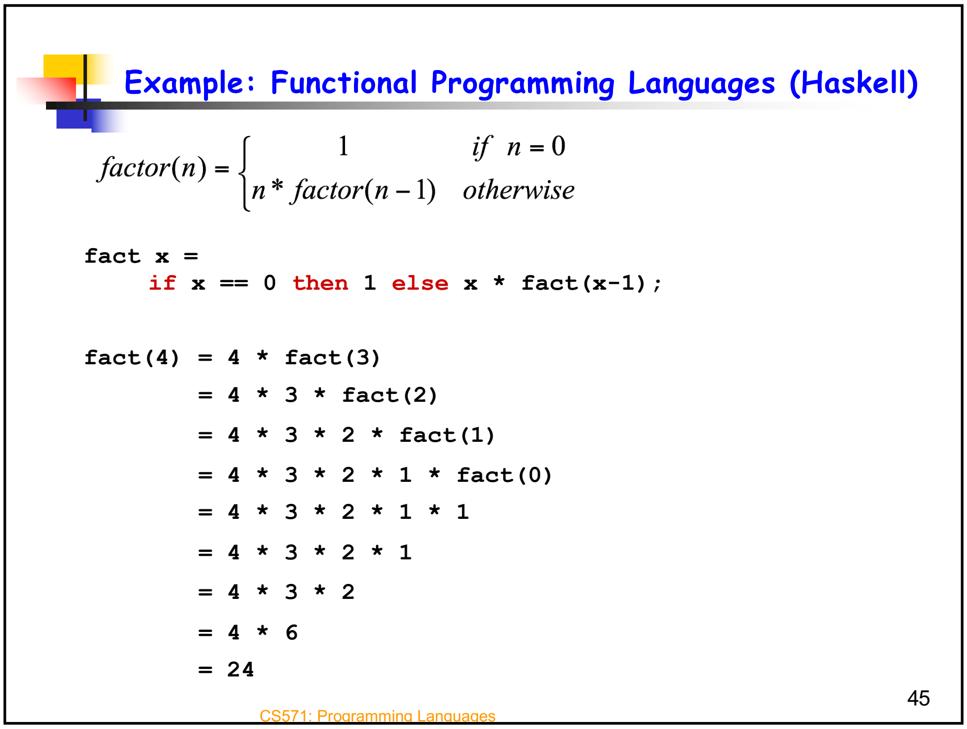
$$factor(n) = \begin{cases} 1 & if \;\; n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$



Example: Functional Programming Languages (Haskell)

$$factor(n) = \begin{cases} 1 & if \;\; n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

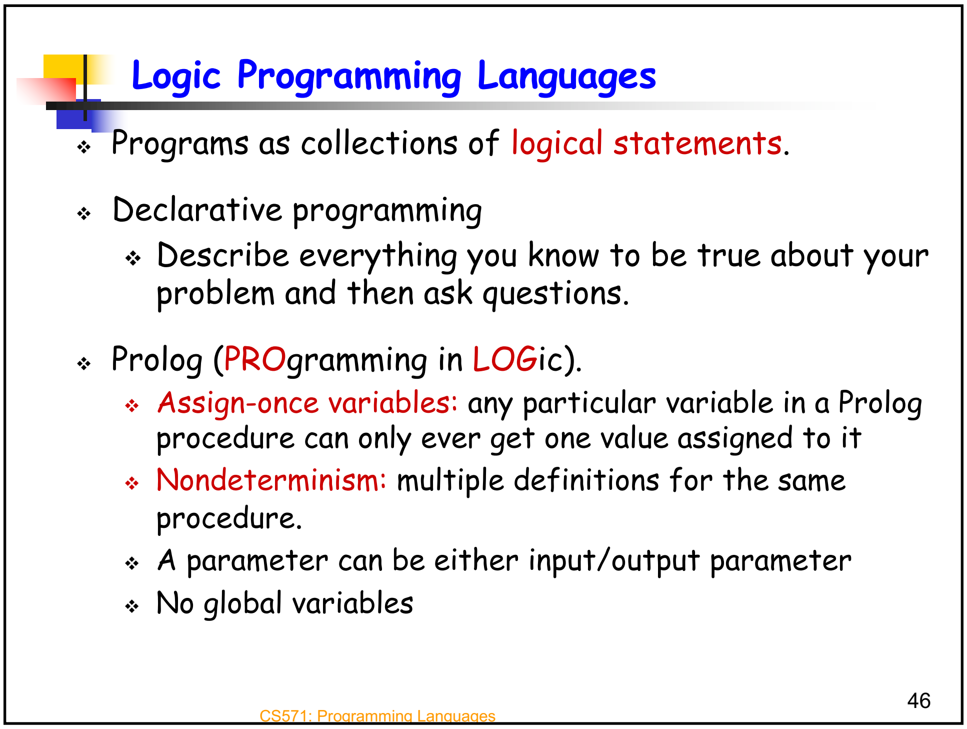
```
fact x =
     if x == 0 then 1 else x * fact(x-1);
```



Example: Functional Programming Languages (Haskell)

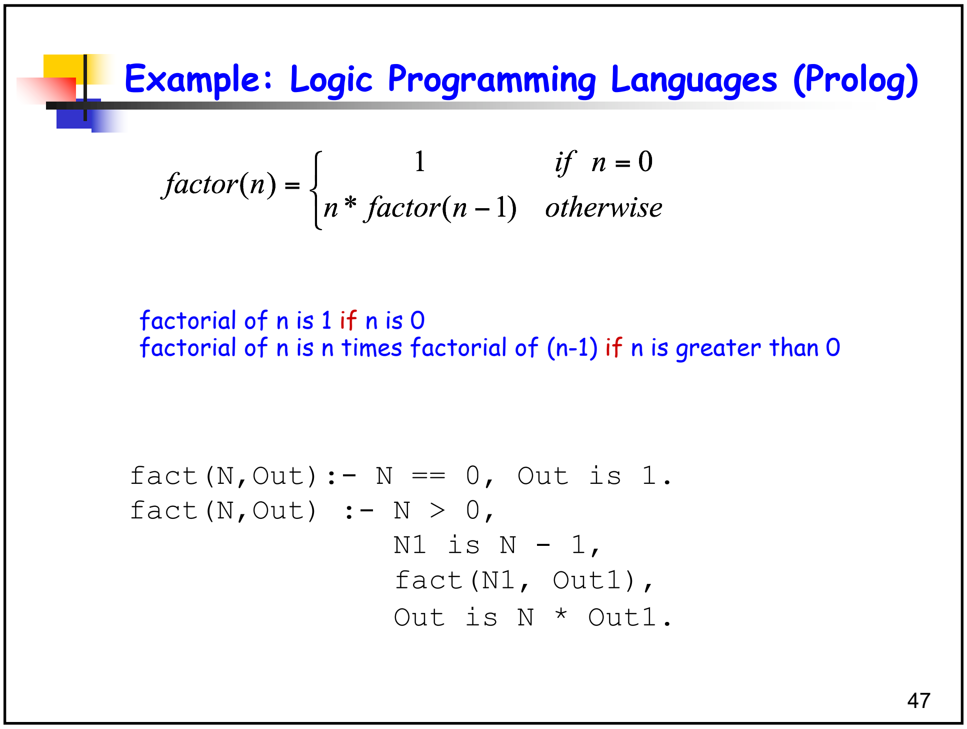
$$factor(n) = \begin{cases} 1 & if \;\; n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

```
fact x =
     if x == 0 then 1 else x * fact(x-1);

fact(4)  = 4 * fact(3)
         = 4 * 3 * fact(2)
         = 4 * 3 * 2 * fact(1)
         = 4 * 3 * 2 * 1 * fact(0)
         = 4 * 3 * 2 * 1 * 1
         = 4 * 3 * 2 * 1
         = 4 * 3 * 2
         = 4 * 6
         = 24
```



Logic Programming Languages

- Programs as collections of logical statements.
- Declarative programming
  - Describe everything you know to be true about your problem and then ask questions.
- Prolog (PROgramming in LOGic).
  - Assign-once variables: any particular variable in a Prolog procedure can only ever get one value assigned to it
  - Nondeterminism: multiple definitions for the same procedure.
  - A parameter can be either input/output parameter
  - No global variables



Example: Logic Programming Languages (Prolog)

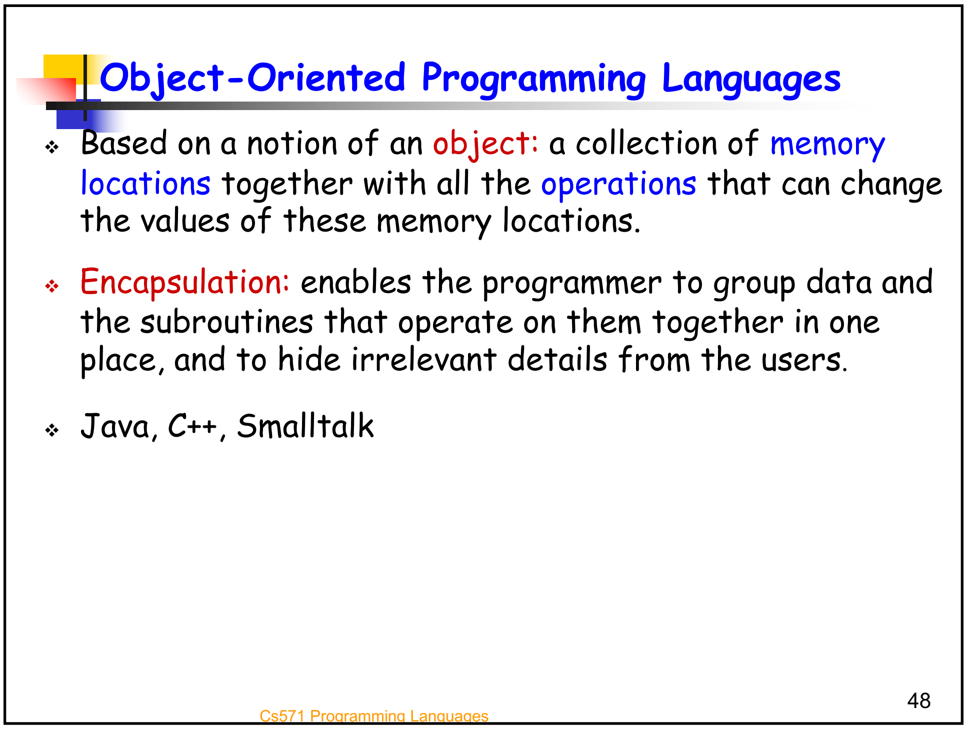
$$factor(n) = \begin{cases} 1 & if \;\; n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

factorial of n is 1 if n is 0
factorial of n is n times factorial of (n-1) if n is greater than 0

```
fact(N,Out):- N == 0, Out is 1.
fact(N,Out) :- N > 0,
               N1 is N - 1,
               fact(N1, Out1),
               Out is N * Out1.
```

47

47



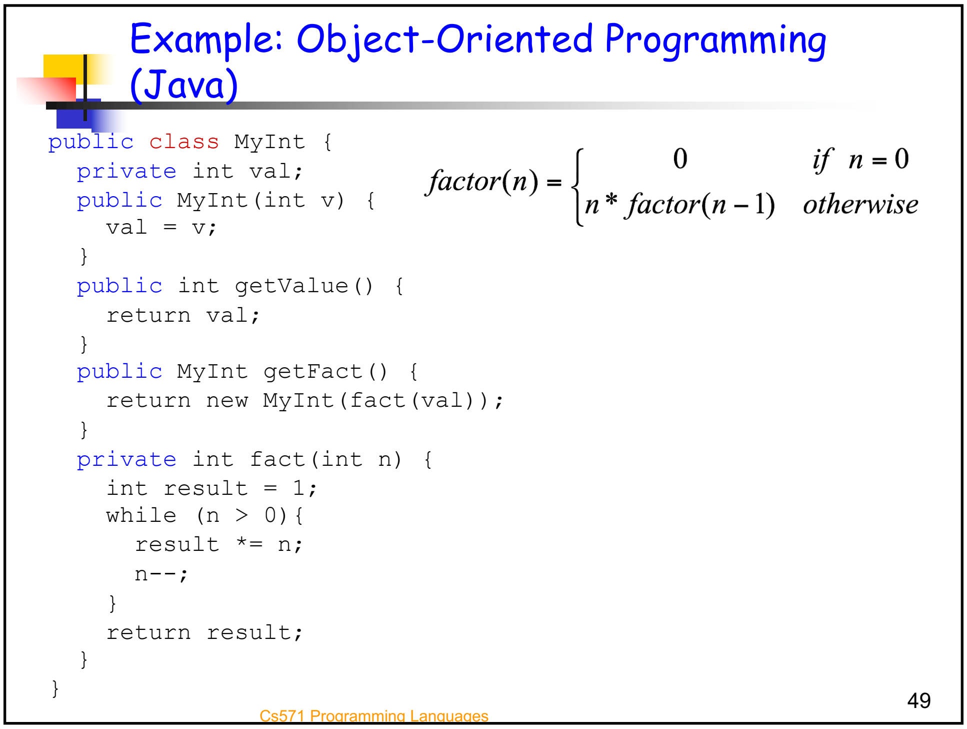
Object-Oriented Programming Languages

- Based on a notion of an object: a collection of memory locations together with all the operations that can change the values of these memory locations.
- Encapsulation: enables the programmer to group data and the subroutines that operate on them together in one place, and to hide irrelevant details from the users.
- Java, C++, Smalltalk

Cs571 Programming Languages

48

48

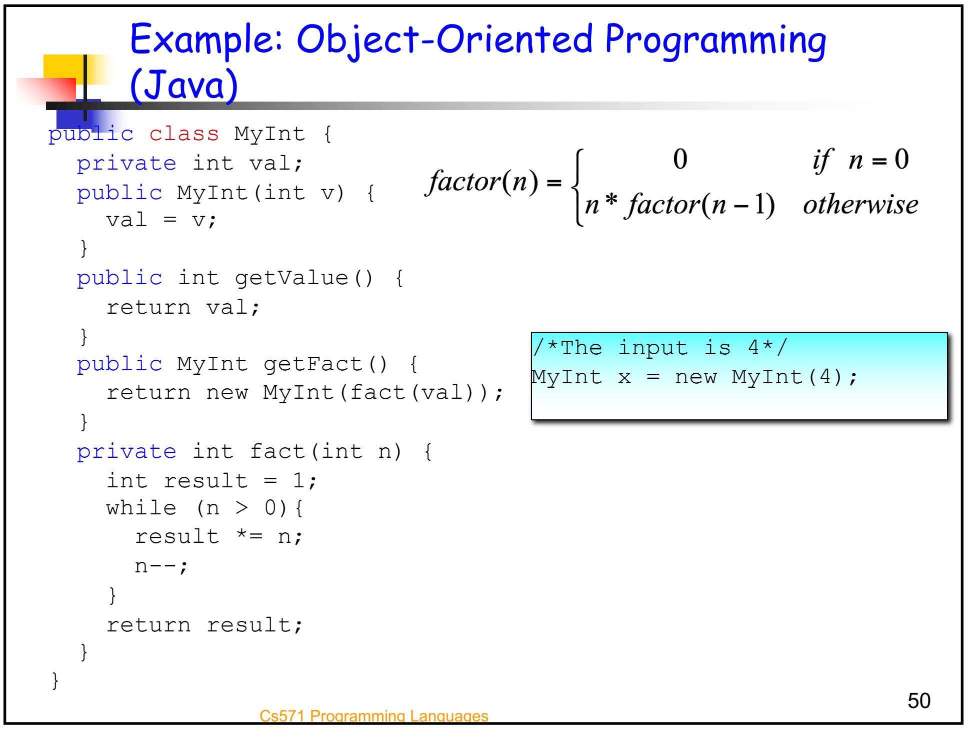


Example: Object-Oriented Programming (Java)

```
public class MyInt {
  private int val;
  public MyInt(int v) {
    val = v;
  }
  public int getValue() {
    return val;
  }
  public MyInt getFact() {
    return new MyInt(fact(val));
  }
  private int fact(int n) {
    int result = 1;
    while (n > 0){
      result *= n;
      n--;
    }
    return result;
  }
}
```

$$factor(n) = \begin{cases} 0 & if\ \ n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

49



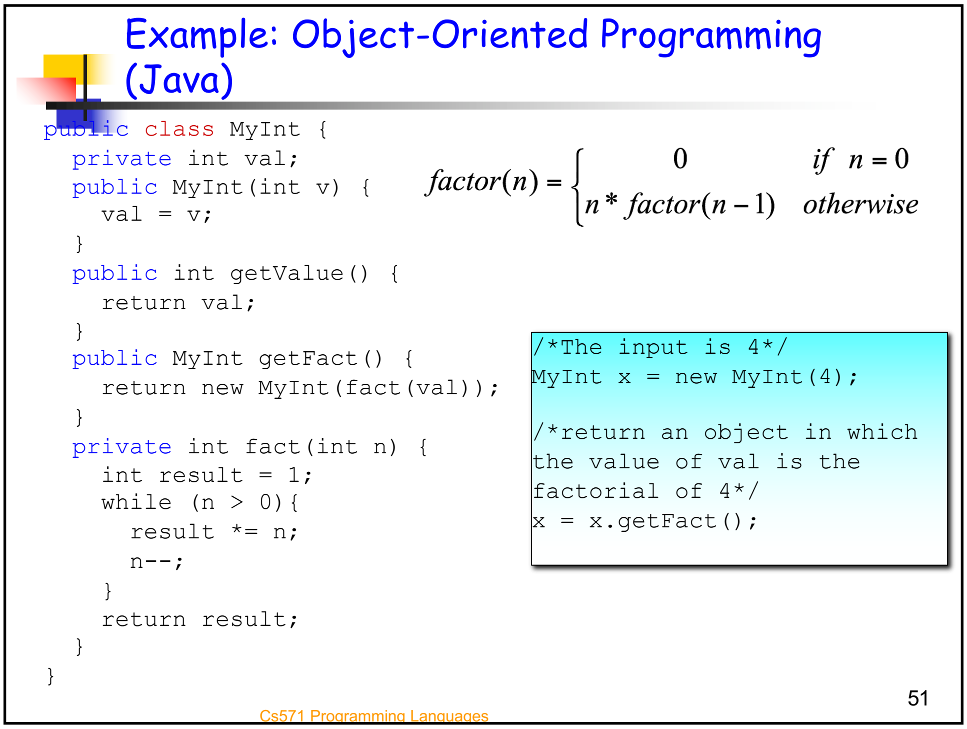
Example: Object-Oriented Programming (Java)

```
public class MyInt {
  private int val;
  public MyInt(int v) {
    val = v;
  }
  public int getValue() {
    return val;
  }
  public MyInt getFact() {
    return new MyInt(fact(val));
  }
  private int fact(int n) {
    int result = 1;
    while (n > 0){
      result *= n;
      n--;
    }
    return result;
  }
}
```

$$factor(n) = \begin{cases} 0 & if\ \ n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

```
/*The input is 4*/
MyInt x = new MyInt(4);
```

50



Example: Object-Oriented Programming (Java)

```
public class MyInt {
  private int val;
  public MyInt(int v) {
    val = v;
  }
  public int getValue() {
    return val;
  }
  public MyInt getFact() {
    return new MyInt(fact(val));
  }
  private int fact(int n) {
    int result = 1;
    while (n > 0){
      result *= n;
      n--;
    }
    return result;
  }
}
```

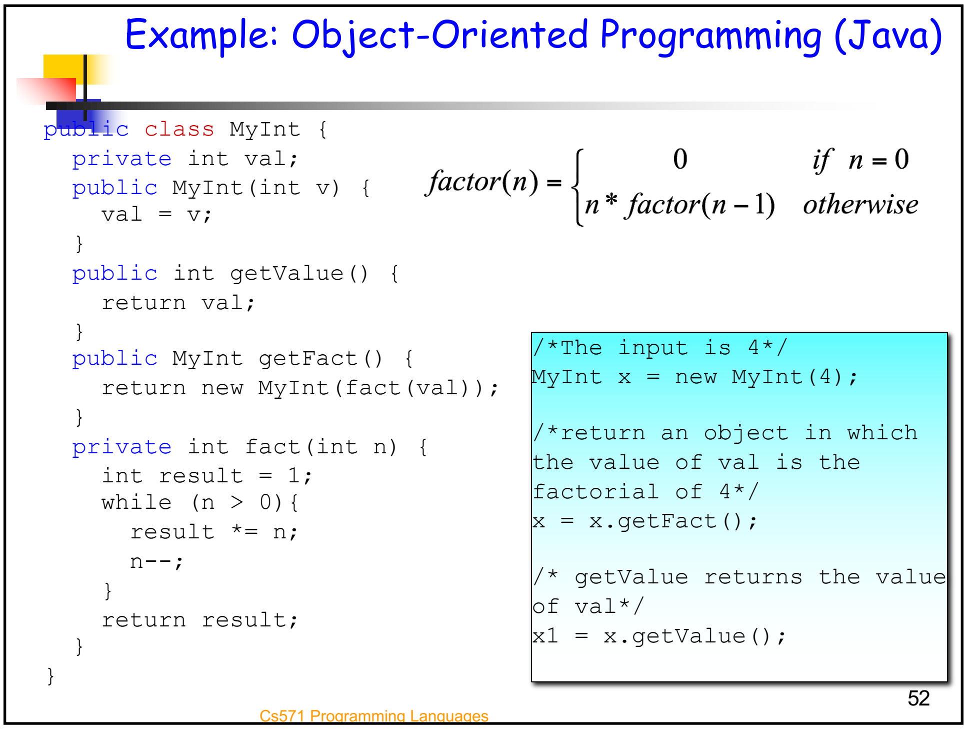
$$factor(n) = \begin{cases} 0 & if\ \ n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

```
/*The input is 4*/
MyInt x = new MyInt(4);

/*return an object in which
the value of val is the
factorial of 4*/
x = x.getFact();
```

Cs571 Programming Languages

51

51



Example: Object-Oriented Programming (Java)

```
public class MyInt {
  private int val;
  public MyInt(int v) {
    val = v;
  }
  public int getValue() {
    return val;
  }
  public MyInt getFact() {
    return new MyInt(fact(val));
  }
  private int fact(int n) {
    int result = 1;
    while (n > 0){
      result *= n;
      n--;
    }
    return result;
  }
}
```

$$factor(n) = \begin{cases} 0 & if\ \ n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

```
/*The input is 4*/
MyInt x = new MyInt(4);

/*return an object in which
the value of val is the
factorial of 4*/
x = x.getFact();

/* getValue returns the value
of val*/
x1 = x.getValue();
```

Cs571 Programming Languages

52

52