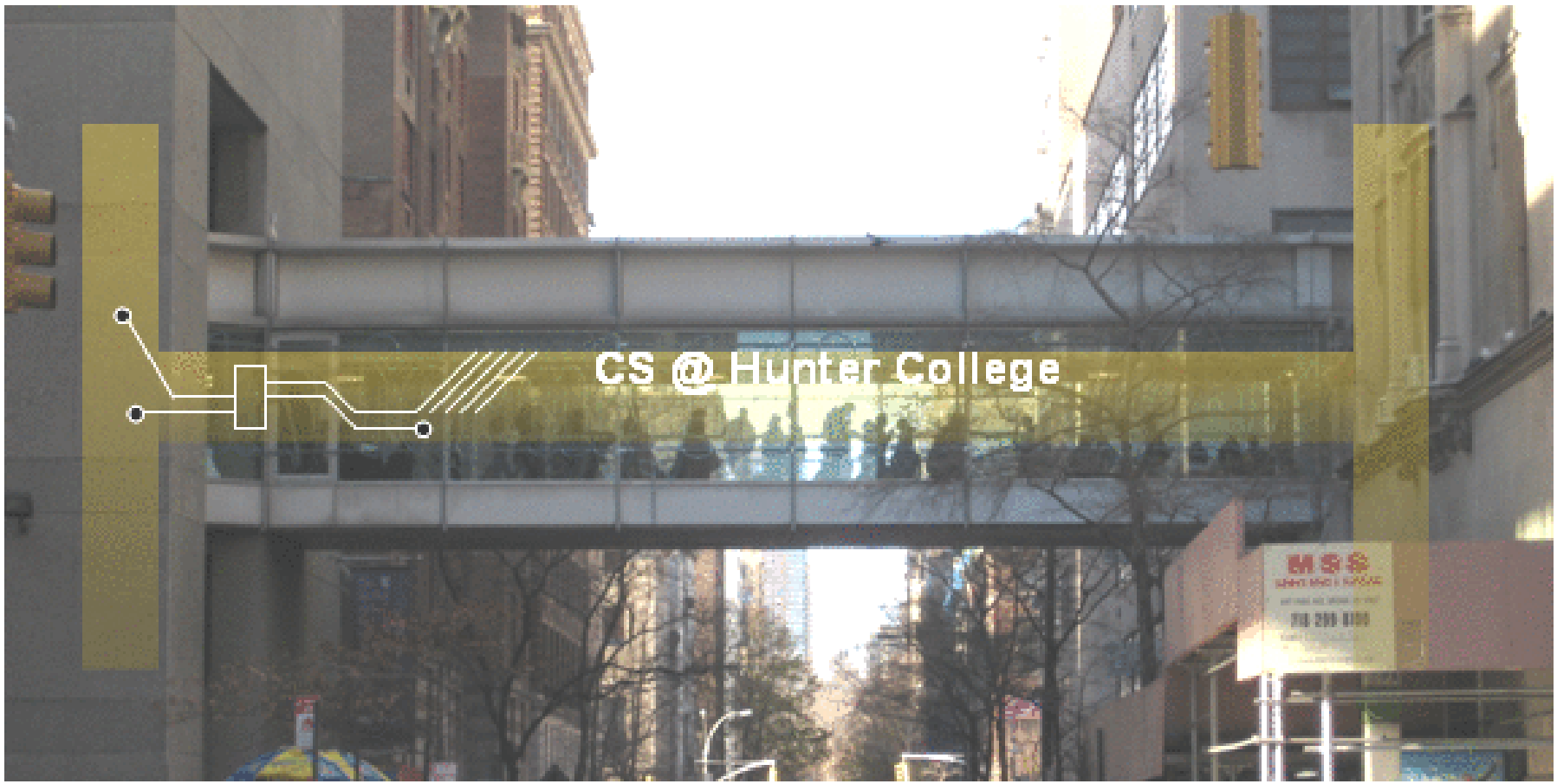
CSCI 127: Introduction to Computer Science



hunter.cuny.edu/csci

Today's Topics

- Recap: Simplified Machine Language
- Recap: Incrementer Design Challenge
- C++: Basic Format & Variables
- I/O and Definite Loops in C++
- More Info on the Final Exam

Today's Topics

- **Recap: Simplified Machine Language**
- Recap: Incrementer Design Challenge
- C++: Basic Format & Variables
- I/O and Definite Loops in C++
- More Info on the Final Exam

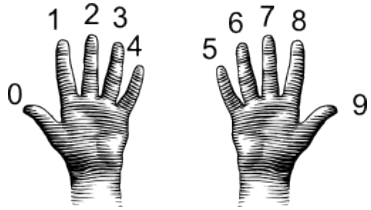
Challenge: What does the code do?

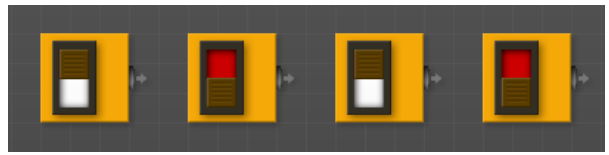
```
ADDI $sp, $sp, -27
ADDI $s3, $zero, 1
ADDI $t0, $zero, 65
ADDI $s2, $zero, 26
SETUP: SB $t0, 0($sp)
ADDI $sp, $sp, 1
SUB $s2, $s2, $s3
ADDI $t0, $t0, 1
BEQ $s2, $zero, DONE
J SETUP
DONE: ADDI $t0, $zero, 0
SB $t0, 0($sp)
ADDI $sp, $sp, -26
ADDI $v0, $zero, 4
ADDI $a0, $sp, 0
syscall
```

Today's Topics

- Recap: Simplified Machine Language
- **Recap: Incrementer Design Challenge**
- C++: Basic Format & Variables
- I/O and Definite Loops in C++
- More Info on the Final Exam

Recap: Design Challenge: Incrementers





- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

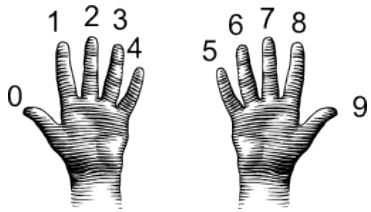
```
def addOne(n):
    m = n+1
    return(m)
```

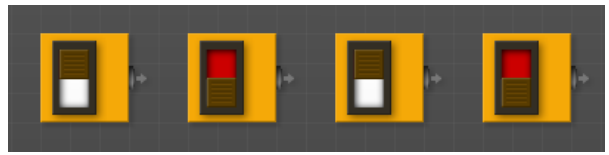
- Challenge: Write an algorithm for incrementing numbers expressed as words.
  Example: `"forty one"` → `"forty two"`

  Hint: Convert to numbers, increment, and convert back to strings.
- Challenge: Write an algorithm for incrementing binary numbers.

  Example: `"1001"` → `"1010"`

Recap: Incrementer Design Challenge



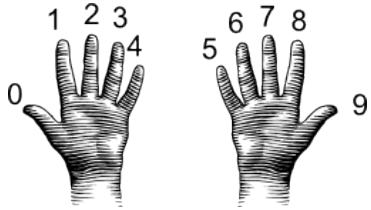


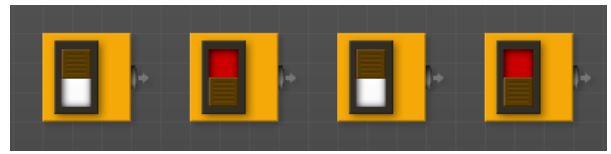
- Challenge: Write an algorithm for incrementing numbers expressed as words. Example: `"forty one"` → `"forty two"`
- Challenge: Write an algorithm for incrementing binary numbers. Example: `"1001"` → `"1010"`
- *Hint: Convert to numbers, increment, and convert back to strings.*

Pseudocode same for both questions:

1. Get user input.
2. Convert to standard decimal number.
3. Add one (increment) the standard decimal number.
4. Convert back to your format.
5. Print the result.

Recap: Incrementer Design Challenge



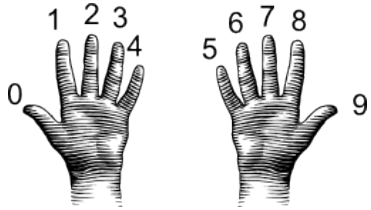


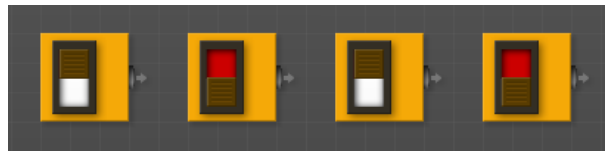
- Challenge: Write an algorithm for incrementing numbers expressed as words. Example: `"forty one"` → `"forty two"`
- Challenge: Write an algorithm for incrementing binary numbers. Example: `"1001"` → `"1010"`

Pseudocode same for both questions:

1. Get user input: `"forty one"`
2. Convert to standard decimal number: `41`
3. Add one (increment) the standard decimal number: `42`
4. Convert back to your format: `"forty two"`
5. Print the result.

Recap: Incrementer Design Challenge



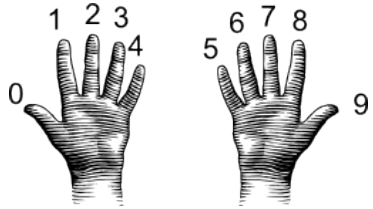


- Challenge: Write an algorithm for incrementing numbers expressed as words. Example: `"forty one"` $\rightarrow$ `"forty two"`
- Challenge: Write an algorithm for incrementing binary numbers. Example: `"1001"` $\rightarrow$ `"1010"`

Pseudocode same for both questions:

1. Get user input: `"1001"`
2. Convert to standard decimal number: 9
3. Add one (increment) the standard decimal number: 10
4. Convert back to your format: `"1010"`
5. Print the result.

Recap: Incrementer Design Challenge

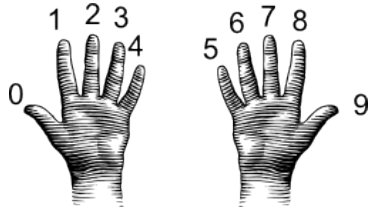


Focus on: Convert to standard decimal number:

```
def convert2Decimal(numString):
    #Start with one-digit numbers:  zero,one,...,nine
    if numString == "zero":
        return(0)
    elif numString == "one":
        return(1)
    elif numString == "two":
        return(2)
    else:
        return(9)
```

Will this work?

Unit Testing: Incrementer Design Challenge



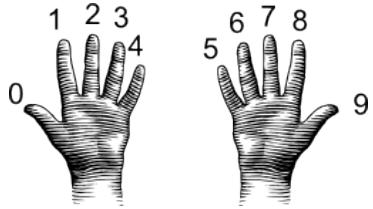
Focus on: Convert to standard decimal number:

```
def convert2Decimal(numString):
    #Start with one-digit numbers:  zero,one,...,nine
    if numString == "zero":
        return(0)
    elif numString == "one":
        return(1)
    elif numString == "two":
        return(2)
    else:
        return(9)
```

Will this work? What inputs would find the error(s)?

Unit Testing: testing individual units/functions/blocks of code to verify correctness. Often automated (e.g. gradescope).

Unit Testing: Incrementer Design Challenge



- Unit Testing: testing individual units/functions/blocks of code to verify correctness. Often automated (e.g. gradescope).
- To test all branches of code, would need to test all inputs: `"zero"`, `"one"`,..., `"nine"`, & some bad inputs.
  Also important to test **edge cases**.
- If large, design automated tests that will "cover" as many branches as possible and use randomly generated inputs:

```
names = ["zero","one",...,"nine"]
x = random.randrange(10)
if x == convert2Decimal(names[x]):
    #PASS
else:
    #FAIL
```

Today's Topics

- Recap: Incrementer Design Challenge
- **C++: Basic Format & Variables**
- I/O and Definite Loops in C++
- More Info on the Final Exam

Challenge:

- *Using what you know from Python, predict what the C++ code will do:*

```
1  //Another C++ program, demonstrating variables
2  #include <iostream>
3  using namespace std;
4
5  int main ()
6  {
7    int year;
8    cout << "Enter a number: ";
9    cin >> year;
10   cout << "Hello " << year << "!!\n\n";
11   return 0;
12 }
```

onlinegdb demo

```cpp
//Another C++ program, demonstrating variables
#include <iostream>
using namespace std;

int main ()
{
  int year;
  cout << "Enter a number: ";
  cin >> year;
  cout << "Hello " << year << "!!\n\n";
  return 0;
}
```

(Demo with `onlinegdb`)

Introduction to C++

```cpp
//Another C++ program, demonstrating variables
#include <iostream>
using namespace std;

int main ()
{
  int year;
  cout << "Enter a number: ";
  cin >> year;
  cout << "Hello " << year << "!!\n\n";
  return 0;
}
```

- C++ is a popular programming language that extends C.
- Fast, efficient, and powerful.
- Used for systems programming (and future courses!).
- Today, we'll introduce the basic structure and simple input/output (I/O) in C/C++.

Introduction to C++

```
1  //Another C++ program, demonstrating variables
2  #include <iostream>
3  using namespace std;
4
5  int main ()
6  {
7    int year;
8    cout << "Enter a number: ";
9    cin >> year;
10   cout << "Hello " << year << "!!\n\n";
11   return 0;
12 }
```

- Programs are organized in functions.

Example:

```
int main()
{
    cout << "Hello world!";
    return(0);
}
```

Introduction to C++

```
1  //Another C++ program, demonstrating variables
2  #include <iostream>
3  using namespace std;
4
5  int main ()
6  {
7    int year;
8    cout << "Enter a number: ";
9    cin >> year;
10   cout << "Hello " << year << "!!\n\n";
11   return 0;
12 }
```

- Programs are organized in functions.
- Variables must be **declared**:
  `int num;`
- Many types available:
  `int, float, char, ...`
- Semicolons separate commands:
  `num = 5; more = 2*num;`
- To print, we'll use `cout <<`:
  `cout << "Hello!!";`
- To get input, we'll use `cin >>`:
  `cin >> num;`
- To use those I/O functions, we put at the top of the program:
  `#include <iostream>`
  `using namespace std;`

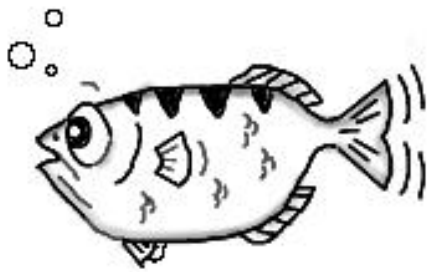
Challenge:

Predict what the following pieces of code will do:

```
//Another C++ program, demonstrating I/O & arithmetic
#include <iostream>
using namespace std;

int main ()
{
  float kg, lbs;
  cout << "Enter kg: ";
  cin >> kg;
  lbs = kg * 2.2;
  cout << endl << "Lbs: " << lbs << "\n\n";
  return 0;
}
```

Side Note: gdb



gdb.org

- Part of Richard Stallman's "GNU is Not Unix" (GNU) project.
- Written in 1986, gdb is the GNU debugger and based on dbx from the Berkeley Distribution of Unix.
- Lightweight, widely-available program that allows you to "step through" your code line-by-line.
- Available on-line (onlinegdb.com) or follow installation instructions in Lab 12.

C++ Demo

```cpp
//Another C++ program, demonstrating I/O & arithmetic
#include <iostream>
using namespace std;

int main ()
{
  float kg, lbs;
  cout << "Enter kg: ";
  cin >> kg;
  lbs = kg * 2.2;
  cout << endl << "Lbs: " << lbs << "\n\n";
  return 0;
}
```

(Demo with `onlinegdb`)

Today's Topics

- Recap: Incrementer Design Challenge
- C++: Basic Format & Variables
- **I/O and Definite Loops in C++**
- More Info on the Final Exam

Challenge:

Predict what the following pieces of code will do:

```
//Another C++ program; Demonstrates loops
#include <iostream>
using namespace std;

int main ()
{
  int i,j;
  for (i = 0; i < 4; i++)
  {
      cout << "The world turned upside down...\n";
  }

  for (j = 10; j > 0; j--)
  {
      cout << j << " ";
  }
  cout << "Blast off!!" << endl;

  return 0;
}
```

C++ Demo

```cpp
//Another C++ program; Demonstrates loops
#include <iostream>
using namespace std;

int main ()
{
  int i,j;
  for (i = 0; i < 4; i++)
  {
     cout << "The world turned upside down...\n";
  }

  for (j = 10; j > 0; j--)
  {
     cout << j << " ";
  }
  cout << "Blast off!!" << endl;

  return 0;
}
```

(Demo with `onlinegdb`)

Definite loops

```
//Another C++ program; Demonstrates loops
#include <iostream>
using namespace std;

int main ()
{
  int i,j;
  for (i = 0; i < 4; i++)
  {
      cout << "The world turned upside down...\n";
  }

  for (j = 10; j > 0; j--)
  {
      cout << j << " ";
  }
  cout << "Blast off!!" << endl;

  return 0;
}
```

General format:

`for` (*initialization* ; *test* ; *updateAction*)
{

command1;
command2;
command3;
...

}

Challenge:

Predict what the following pieces of code will do:

```
//Another C++ program; Demonstrates loops
#include <iostream>
using namespace std;

int main ()
{
  int i,j,size;
  cout << "Enter size: ";
  cin >> size;
  for (i = 0; i < size; i++)
  {
    for (j = 0; j < size; j++)
      cout << "*";
    cout << endl;
  }
  cout << "\n\n";
  for (i = size; i > 0; i--)
  {
    for (j = 0; j < i; j++)
      cout << "*";
    cout << endl;
  }
  return 0;
}
```

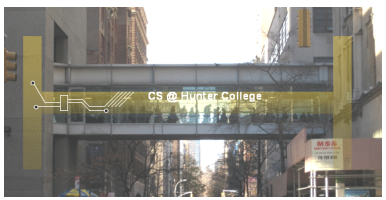
C++ Demo

```cpp
//Another C++ program; Demonstrates loops
#include <iostream>
using namespace std;

int main ()
{
  int i,j,size;
  cout << "Enter size: ";
  cin >> size;
  for (i = 0; i < size; i++)
  {
    for (j = 0; j < size; j++)
      cout << "*";
    cout << endl;
  }
  cout << "\n\n";
  for (i = size; i > 0; i--)
  {
    for (j = 0; j < i; j++)
      cout << "*";
    cout << endl;
  }
  return 0;
}
```

(Demo with `onlinegdb`)

Recap: C++



- C++ is a popular programming language that extends C.
- Input/Output (I/O):
  - `cin >>`
  - `cout <<`
- Definite loops:

```
for (i = 0; i < 10; i++) {
    ...
}
```

Today's Topics

- Recap: Incrementer Design Challenge
- C++: Basic Format & Variables
- I/O and Definite Loops in C++
- **More Info on the Final Exam**

Final Overview: Format

- Closed book. No electronic devices allowed. If we see your phone we will take it until the end of the exam.
- You can have 1 piece of **8.5" x 11"** paper.
  - With notes, examples, programs: what will help you on the exam.
  - Best if you design/write yours, it's excellent way to study.
- The exam format:
  - Same format as past exams posted on course website
  - Questions based on course topics, and are variations on the programming assignments, lab exercises, and lecture design challenges.
  - Style of questions: short answer, fill in the program (one line of code per box), multiple choice, select all, replace value, modify program, translate & write complete programs.
- Past exams available on webpage (includes answer keys).

How to Prepare

- Emphasis of this course is on analytic reasoning and problem solving.
- The best way to prepare to do problems (reading & watching videos can clarify but not replace problem solving).
- Repeat, while there are past exams:
  - Choose a past exam (see webpage).
  - With only a note sheet, work through in 1 hour (half the time).
  - Grade yourself (answers on webpage).
  - Ask about those that don't make sense.
  - Rewrite answers & organize by type/question number.
  - Adjust/rewrite note sheet to include what you wished you had.
- Aim to complete 7 to 10 past exams (one a day in the week leading up to the final).

Final Overview: Rules

You will get credit for you answers **only if**:

- Your answer uses language constructs that were covered in the course.
- Your answer is not oddly identically to that of another student or is the answer for another version of the exam.

All acts of academic dishonesty will be reported to the Office of Academic and Student Affairs and will result in a 0 grade on the exam.

Final Exam Practice Rounds:

For each question, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- **Write a function that takes a weight in kilograms and returns the weight in pounds.**

```
def kg2lbs(kg):
    ...
    return(lbs)
```

Final Exam Practice Rounds:

For each question, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- **Write a function that takes a weight in kilograms and returns the weight in pounds.**

```
def kg2lbs(kg)
    lbs = kg * 2.2
    return(lbs)
```

Final Exam Practice Rounds:

For each question, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- **Write a function that takes a string and returns its length.**

```
def sLength(str):
    ...
    return(length)
```

Final Exam Practice Rounds:

For each question, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- **Write a function that takes a string and returns its length.**

```
def sLength(str):
    length = len(str)
    return(length)
```

Final Exam Practice Rounds:

For each question, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- **Write a function that, given a DataFrame, returns the minimal value in the "Manhattan" column.**

```
def getMin(df):
    ...
    return(min)
```

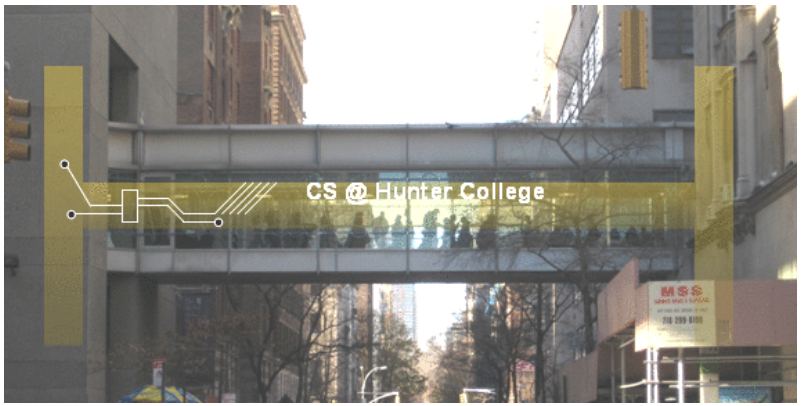
Final Exam Practice Rounds:

For each question below, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- **Write a function that, given a DataFrame, returns the minimal value in the "Manhattan" column.**

```
def getMin(df):
    min = df["Manhattan"].min()
    return(min)
```

Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Schedule an appointment to take the Quiz in lab 1001G Hunter North
- If you haven't already, schedule an appointment to take the Code Review in lab 1001G Hunter North
- Submit this week's programming assignments
- If you need help, schedule an appointment for Tutoring in lab
- Take the Lecture Preview on Blackboard on Monday (or no later than 10am on Tuesday)

Lecture Slips & Writing Boards

- Hand your lecture slip to a UTA.
- Return writing boards as you leave.