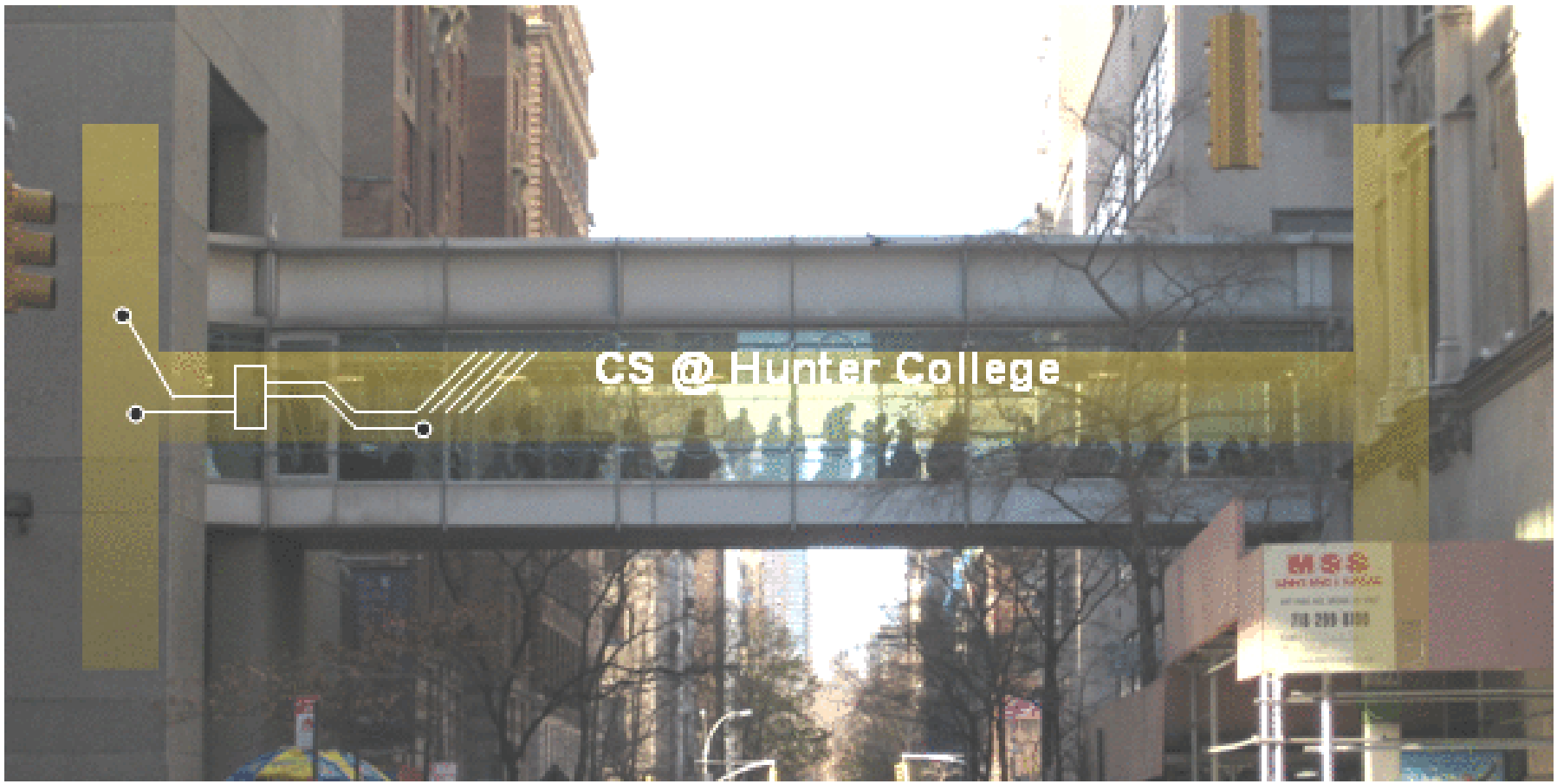
CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

Announcements

- Please always read all Blackboard announcements

Announcements

- Please always read all Blackboard announcements
- Online help is available in multiple forms when school is in session:

Announcements

- Please always read all Blackboard announcements
- Online help is available in multiple forms when school is in session:
  - **Email**: csci127help@gmail.com

Announcements

- Please always read all Blackboard announcements
- Online help is available in multiple forms when school is in session:
  - **Email**: csci127help@gmail.com
  - **Discussion Board**: on Blackboard, link on purple menu bar

Announcements

- Please always read all Blackboard announcements
- Online help is available in multiple forms when school is in session:
  - **Email**: csci127help@gmail.com
  - **Discussion Board**: on Blackboard, link on purple menu bar
  - **Drop-in tutoring (12pm-5pm)**:

Announcements

- Please always read all Blackboard announcements
- Online help is available in multiple forms when school is in session:
  - **Email**: csci127help@gmail.com
  - **Discussion Board**: on Blackboard, link on purple menu bar
  - **Drop-in tutoring (12pm-5pm)**: sign in here: https://bit.ly/csci127Tutoring then join the session here: https://bit.ly/csci127TutoringSession

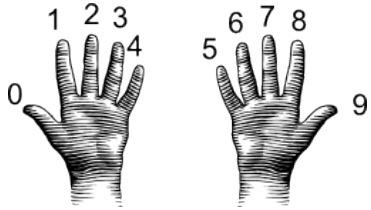
Today's Topics

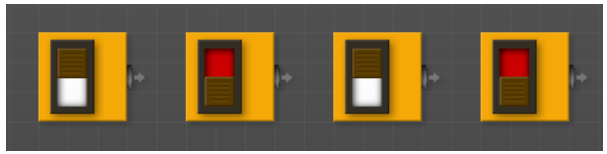
- Recap: Incrementer Design Challenge
- C++: Basic Format & Variables
- I/O and Definite Loops in C++
- More Info on the Final Exam

Today's Topics

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

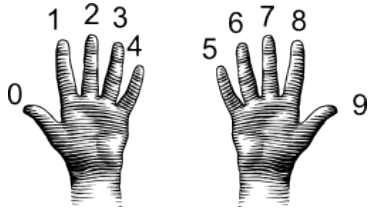
Recap: Design Challenge: Incrementers

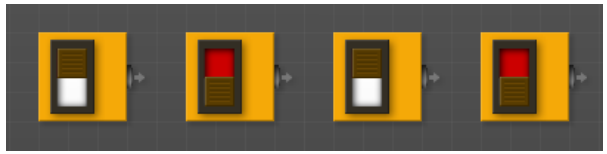




- Simplest arithmetic: add one ("increment") a variable.

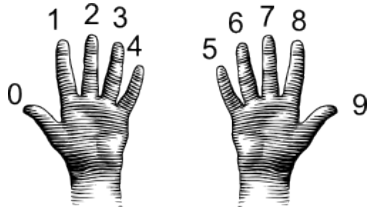
Recap: Design Challenge: Incrementers

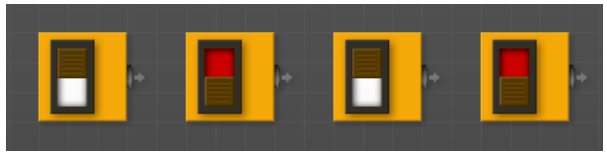




- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

Recap: Design Challenge: Incrementers

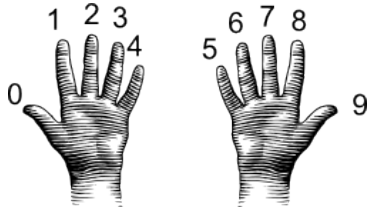


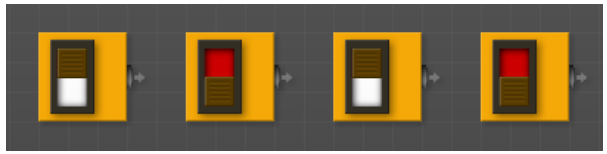


- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

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

Recap: Design Challenge: Incrementers



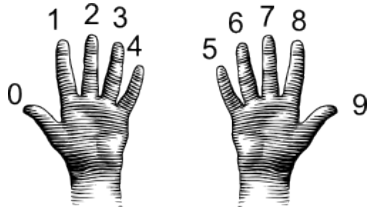


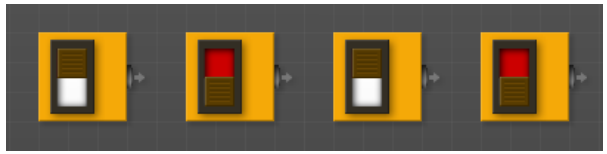
- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

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

- Challenge: Write an algorithm for incrementing numbers expressed as words.

Recap: Design Challenge: Incrementers



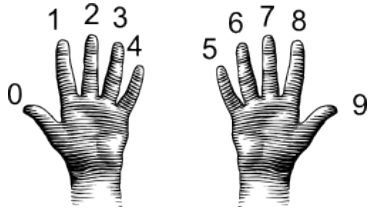


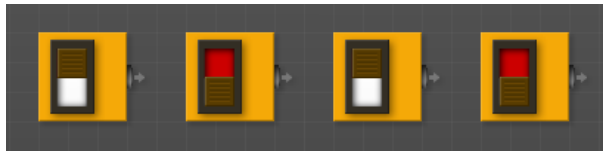
- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

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

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

Recap: Design Challenge: Incrementers





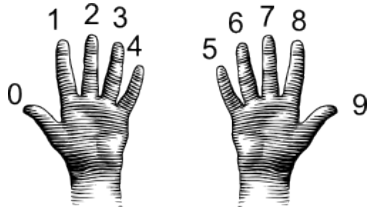
- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

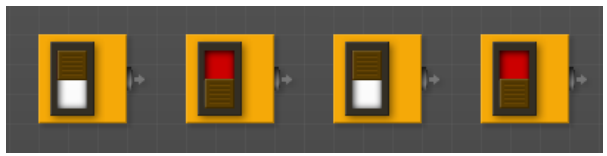
```
def addOne(n):
    m = n+1
    return(m)
```

- 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.

Recap: Design Challenge: Incrementers





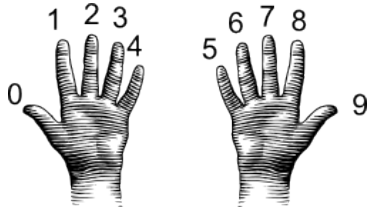
- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

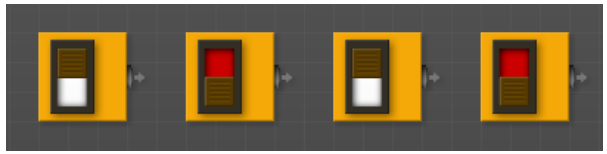
```
def addOne(n):
    m = n+1
    return(m)
```

- 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.

Recap: Design Challenge: Incrementers





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

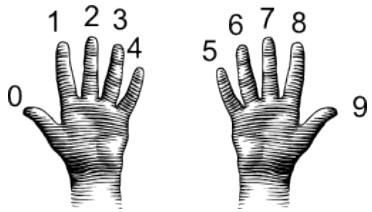
```
def addOne(n):
    m = n+1
    return(m)
```

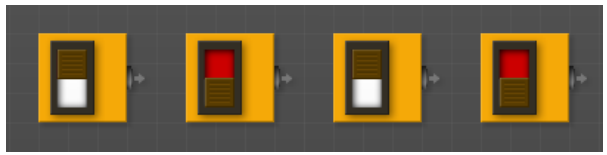
- Challenge: Write an algorithm for incrementing numbers expressed as words.
  Example: `"forty one"` → `"forty two"`

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

  Example: `"1001"` → `"1010"`

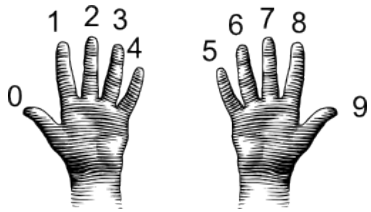
Recap: Incrementer Design Challenge

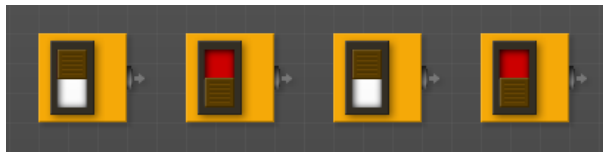




- Challenge: Write an algorithm for incrementing numbers expressed as words. Example: `"forty one"` $\rightarrow$ `"forty two"`

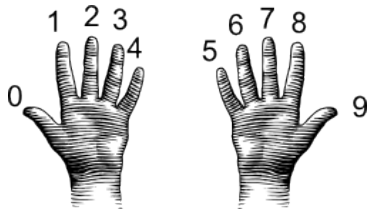
Recap: Incrementer Design Challenge

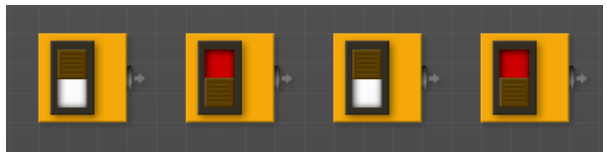




- Challenge: Write an algorithm for incrementing numbers expressed as words. Example: `"forty one"` $\rightarrow$ `"forty two"`
- Challenge: Write an algorithm for incrementing binary numbers. Example: `"1001"` $\rightarrow$ `"1010"`
- *Hint: Convert to numbers, increment, and convert back to strings.*

Recap: Incrementer Design Challenge



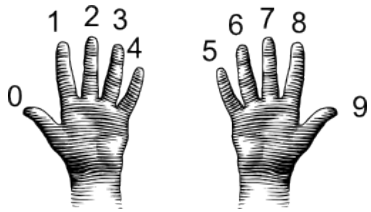


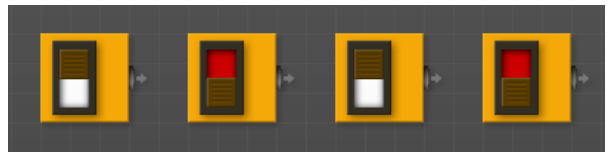
- Challenge: Write an algorithm for incrementing numbers expressed as words. Example: `"forty one"` $\rightarrow$ `"forty two"`
- Challenge: Write an algorithm for incrementing binary numbers. Example: `"1001"` $\rightarrow$ `"1010"`
- *Hint: Convert to numbers, increment, and convert back to strings.*

Pseudocode same for both questions:

1. Get user input.

Recap: Incrementer Design Challenge



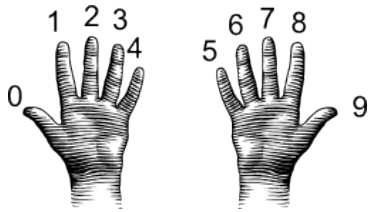


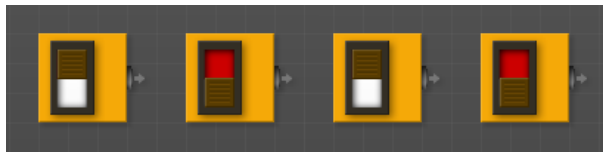
- Challenge: Write an algorithm for incrementing numbers expressed as words. Example: `"forty one"` $\rightarrow$ `"forty two"`
- Challenge: Write an algorithm for incrementing binary numbers. Example: `"1001"` $\rightarrow$ `"1010"`
- *Hint: Convert to numbers, increment, and convert back to strings.*

Pseudocode same for both questions:

1. Get user input.
2. Convert to standard decimal number.

Recap: Incrementer Design Challenge



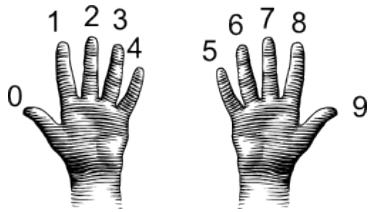


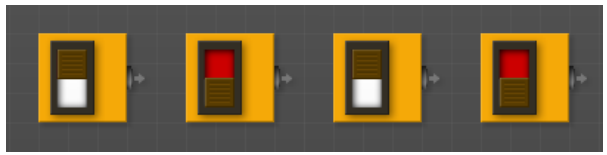
- Challenge: Write an algorithm for incrementing numbers expressed as words. Example: `"forty one"` $\rightarrow$ `"forty two"`
- Challenge: Write an algorithm for incrementing binary numbers. Example: `"1001"` $\rightarrow$ `"1010"`
- *Hint: Convert to numbers, increment, and convert back to strings.*

Pseudocode same for both questions:

1. Get user input.
2. Convert to standard decimal number.
3. Add one (increment) the standard decimal number.

Recap: Incrementer Design Challenge



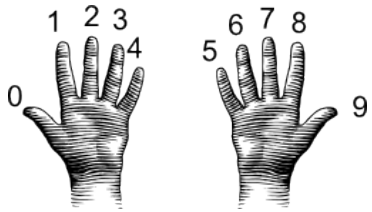


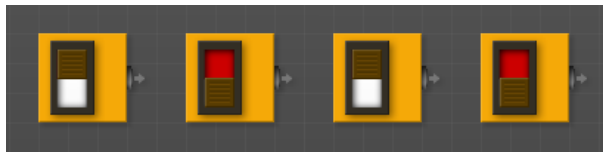
- Challenge: Write an algorithm for incrementing numbers expressed as words. Example: `"forty one"` $\rightarrow$ `"forty two"`
- Challenge: Write an algorithm for incrementing binary numbers. Example: `"1001"` $\rightarrow$ `"1010"`
- *Hint: Convert to numbers, increment, and convert back to strings.*

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.

Recap: Incrementer Design Challenge



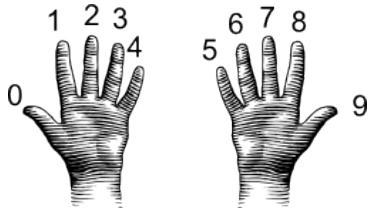


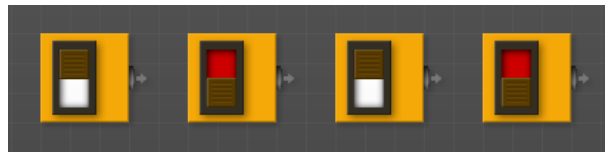
- Challenge: Write an algorithm for incrementing numbers expressed as words. Example: `"forty one"` → `"forty two"`
- Challenge: Write an algorithm for incrementing binary numbers. Example: `"1001"` → `"1010"`
- *Hint: Convert to numbers, increment, and convert back to strings.*

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



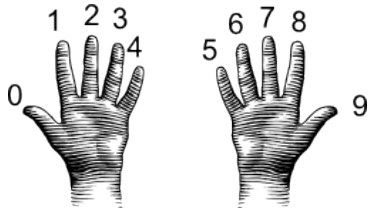


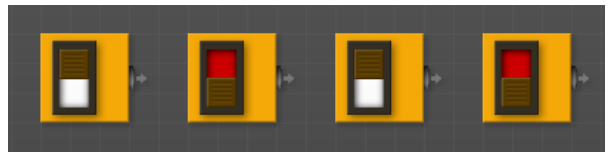
- Challenge: Write an algorithm for incrementing numbers expressed as words. Example: `"forty one"` $\rightarrow$ `"forty two"`
- Challenge: Write an algorithm for incrementing binary numbers. Example: `"1001"` $\rightarrow$ `"1010"`

Pseudocode same for both questions:

1. Get user input: `"forty one"`

Recap: Incrementer Design Challenge



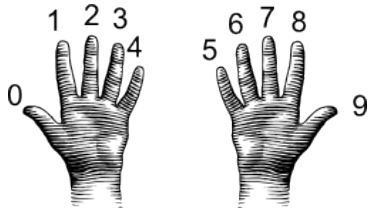


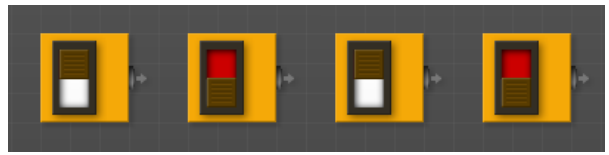
- Challenge: Write an algorithm for incrementing numbers expressed as words. Example: `"forty one"` $\rightarrow$ `"forty two"`
- Challenge: Write an algorithm for incrementing binary numbers. Example: `"1001"` $\rightarrow$ `"1010"`

Pseudocode same for both questions:

1. Get user input: `"forty one"`
2. Convert to standard decimal number: `41`

Recap: Incrementer Design Challenge



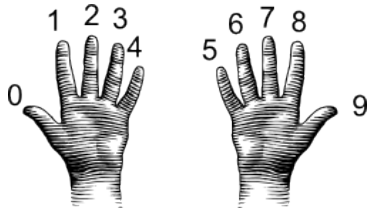


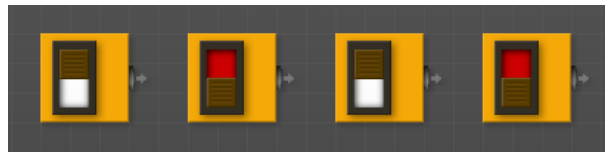
- Challenge: Write an algorithm for incrementing numbers expressed as words. Example: `"forty one"` $\rightarrow$ `"forty two"`
- Challenge: Write an algorithm for incrementing binary numbers. Example: `"1001"` $\rightarrow$ `"1010"`

Pseudocode same for both questions:

1. Get user input: `"forty one"`
2. Convert to standard decimal number: `41`
3. Add one (increment) the standard decimal number: `42`

Recap: Incrementer Design Challenge



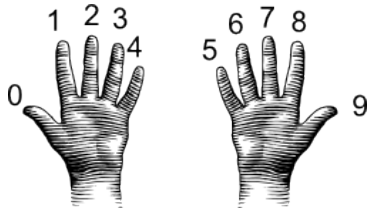


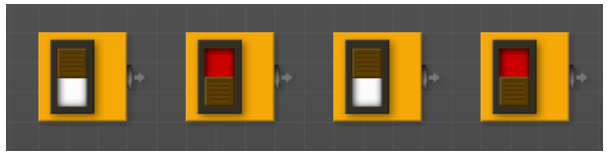
- Challenge: Write an algorithm for incrementing numbers expressed as words. Example: `"forty one"` $\rightarrow$ `"forty two"`
- Challenge: Write an algorithm for incrementing binary numbers. Example: `"1001"` $\rightarrow$ `"1010"`

Pseudocode same for both questions:

1. Get user input: `"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"`

Recap: Incrementer Design Challenge



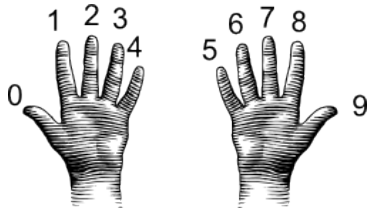


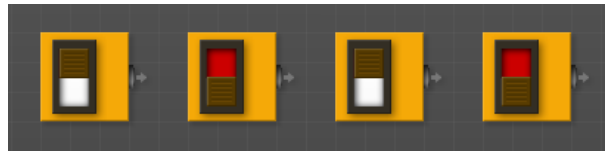
- Challenge: Write an algorithm for incrementing numbers expressed as words. Example: `"forty one"` $\rightarrow$ `"forty two"`
- Challenge: Write an algorithm for incrementing binary numbers. Example: `"1001"` $\rightarrow$ `"1010"`

Pseudocode same for both questions:

1. Get user input: `"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



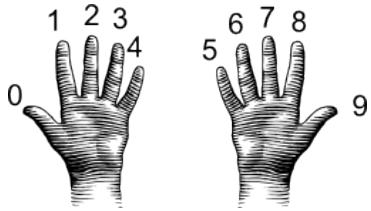


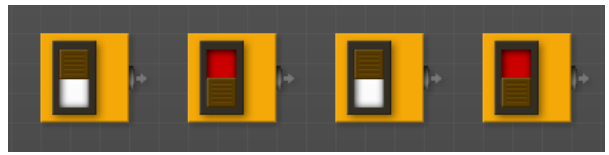
- Challenge: Write an algorithm for incrementing numbers expressed as words. Example: `"forty one"` $\rightarrow$ `"forty two"`
- Challenge: Write an algorithm for incrementing binary numbers. Example: `"1001"` $\rightarrow$ `"1010"`

Pseudocode same for both questions:

1. Get user input: `"1001"`

Recap: Incrementer Design Challenge



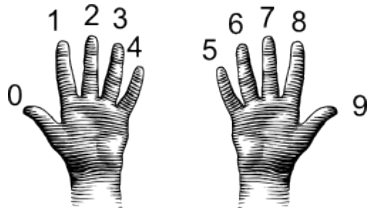


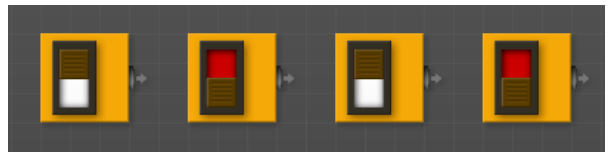
- Challenge: Write an algorithm for incrementing numbers expressed as words. Example: `"forty one"` $\rightarrow$ `"forty two"`
- Challenge: Write an algorithm for incrementing binary numbers. Example: `"1001"` $\rightarrow$ `"1010"`

Pseudocode same for both questions:

1. Get user input: `"1001"`
2. Convert to standard decimal number: 9

Recap: Incrementer Design Challenge



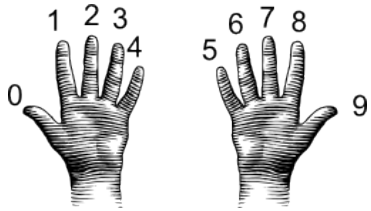


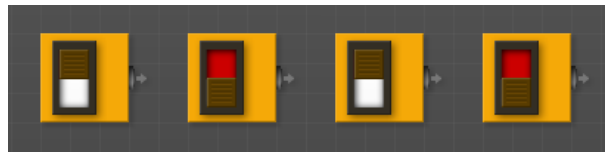
- Challenge: Write an algorithm for incrementing numbers expressed as words. Example: `"forty one"` $\rightarrow$ `"forty two"`
- Challenge: Write an algorithm for incrementing binary numbers. Example: `"1001"` $\rightarrow$ `"1010"`

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

Recap: Incrementer Design Challenge



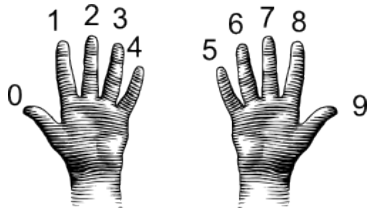


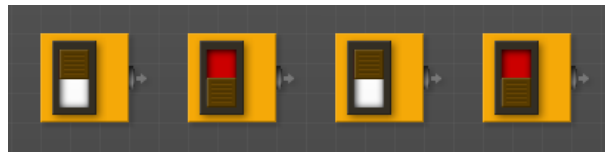
- Challenge: Write an algorithm for incrementing numbers expressed as words. Example: `"forty one"` $\rightarrow$ `"forty two"`
- Challenge: Write an algorithm for incrementing binary numbers. Example: `"1001"` $\rightarrow$ `"1010"`

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"`

Recap: Incrementer Design Challenge



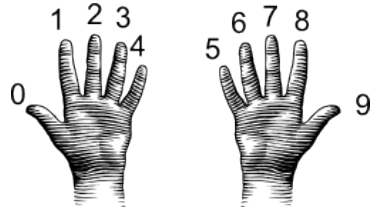


- Challenge: Write an algorithm for incrementing numbers expressed as words. Example: `"forty one"` $\rightarrow$ `"forty two"`
- Challenge: Write an algorithm for incrementing binary numbers. Example: `"1001"` $\rightarrow$ `"1010"`

Pseudocode same for both questions:

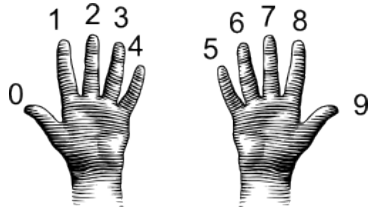
1. Get user input: `"1001"`
2. Convert to standard decimal number: 9
3. Add one (increment) the standard decimal number: 10
4. Convert back to your format: `"1010"`
5. Print the result.

Recap: Incrementer Design Challenge



Focus on: Convert to standard decimal number:

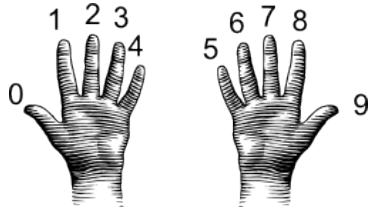
Recap: Incrementer Design Challenge



Focus on: Convert to standard decimal number:

```
def convert2Decimal(numString):
```

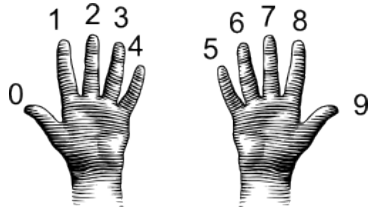
Recap: Incrementer Design Challenge



Focus on: Convert to standard decimal number:

```
def convert2Decimal(numString):
    #Start with one-digit numbers:  zero,one,...,nine
```

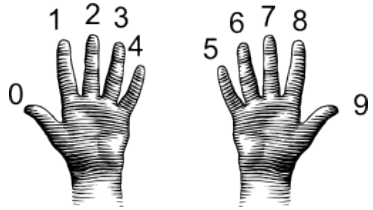
Recap: Incrementer Design Challenge



Focus on: Convert to standard decimal number:

```
def convert2Decimal(numString):
    #Start with one-digit numbers:  zero,one,...,nine
    if numString == "zero":
        return(0)
```

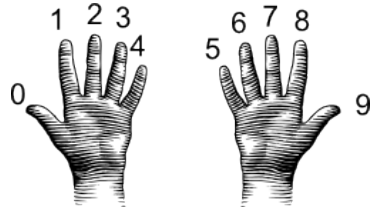
Recap: Incrementer Design Challenge



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)
```

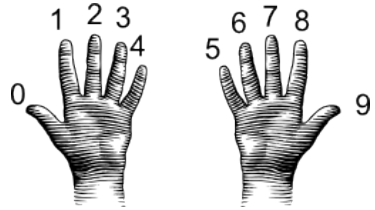
Recap: Incrementer Design Challenge



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(1)
```

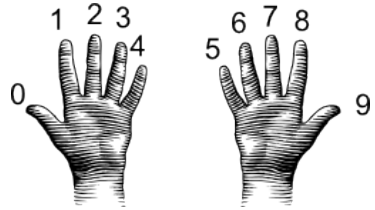
Recap: Incrementer Design Challenge



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(1)
    else:
        return(9)
```

Recap: Incrementer Design Challenge

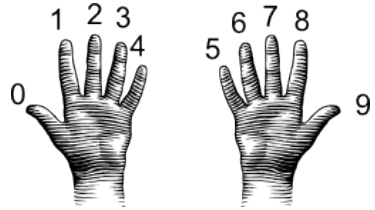


Focus on: Convert to standard decimal number:

```
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(1)
    else:
        return(9)
```

Will this work?

Recap: Incrementer Design Challenge

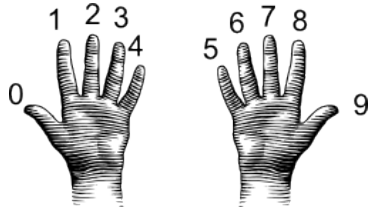


Focus on: Convert to standard decimal number:

```
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(1)
    else:
        return(9)
```

Will this work?

Unit Testing: Incrementer Design Challenge

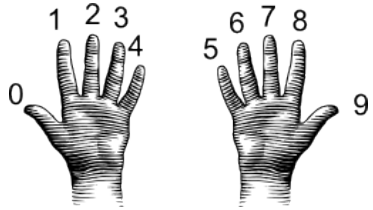


Focus on: Convert to standard decimal number:

```
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(1)
    else:
        return(9)
```

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

Unit Testing: Incrementer Design Challenge



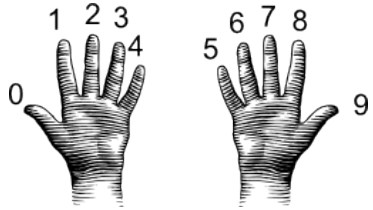
Focus on: Convert to standard decimal number:

```
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(1)
    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.

Unit Testing: Incrementer Design Challenge



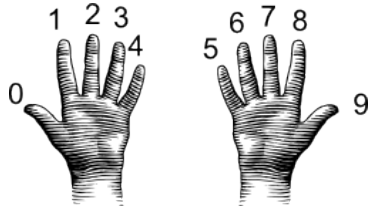
Focus on: Convert to standard decimal number:

```
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(1)
    else:
        return(9)
```

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

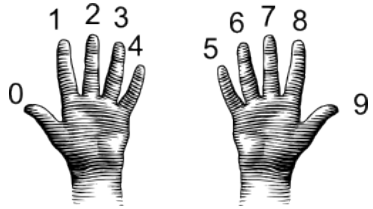
Unit Testing: testing individual units/functions/blocks of code to verify correctness. Often automated (e.g. gradescope).

Unit Testing: Incrementer Design Challenge



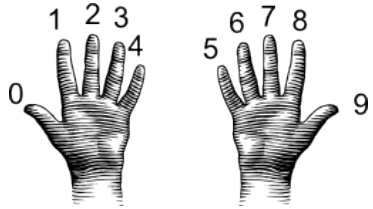
- Unit Testing: testing individual units/functions/blocks of code to verify correctness. Often automated (e.g. gradescope).
- To test all branches of code, would need to test all inputs: `"zero"`, `"one"`,..., `"nine"`, & some bad inputs.

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. Often do, if important or small.
- If large, design automated tests that will "cover" as many branches as possible and use randomly generated inputs:

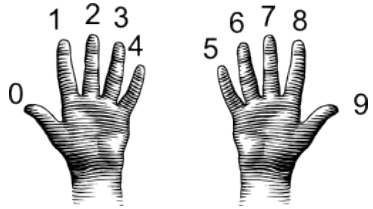
Unit Testing: Incrementer Design Challenge



- 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. Often do, if important or small.
- If large, design automated tests that will "cover" as many branches as possible and use randomly generated inputs:

```
names = ["zero","one",...,"nine"]
```

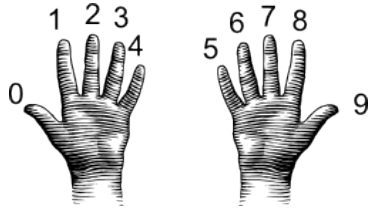
Unit Testing: Incrementer Design Challenge



- 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. Often do, if important or small.
- 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)
```

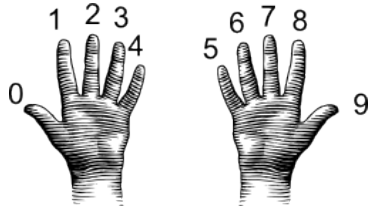
Unit Testing: Incrementer Design Challenge



- 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. Often do, if important or small.
- 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]):
```

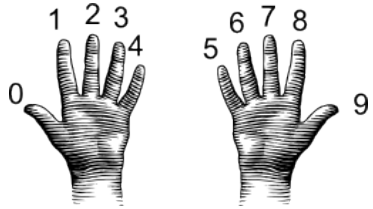
Unit Testing: Incrementer Design Challenge



- 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. Often do, if important or small.
- 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
```

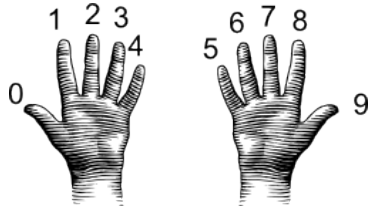
Unit Testing: Incrementer Design Challenge



- 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. Often do, if important or small.
- 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:
```

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. Often do, if important or small.
- 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++

```
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 }
```

- C++ is a popular programming language that extends 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 }
```

- C++ is a popular programming language that extends C.
- Fast, efficient, and powerful.

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!).

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++

- Programs are organized in functions.

```
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 }
```

Introduction to C++

- Programs are organized in functions.

Example:

```
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 }
```

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()`

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()
{
```

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++

- Programs are organized in functions.

```
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 }
```

Introduction to C++

- Programs are organized in functions.
- Variables must be **declared**:

```
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 }
```

Introduction to C++

- Programs are organized in functions.
- Variables must be **declared**:
  `int num;`

```
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 }
```

Introduction to C++

- Programs are organized in functions.
- Variables must be **declared**:
  `int num;`
- Many types available:
  `int, float, char, ...`

```
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 }
```

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:

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;`

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 <<`:

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!!";`

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 >>`:

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;`

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:

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 }
```

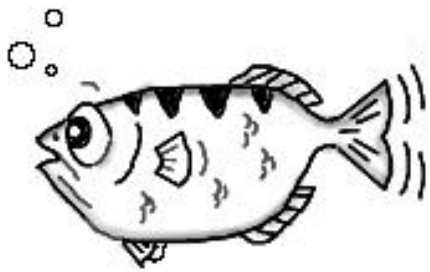
- Programs are organized in functions.
- Variables must be **declared**:
  `int num;`
- Many types available:
  `int, float, char, ...`
- Semicolons separate commands:
  `num = 5; more = 2*num;`
- To print, we'll use `cout <<`:
  `cout << "Hello!!";`
- To get input, we'll use `cin >>`:
  `cin >> num;`
- 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;
}
```

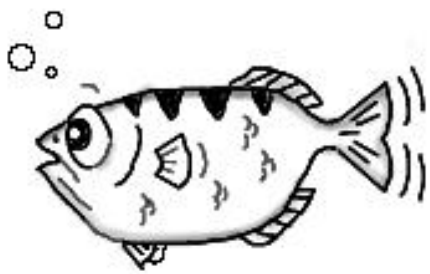
Side Note: gdb



gdb.org

- Part of Richard Stallman's "GNU is Not Unix" (GNU) project.

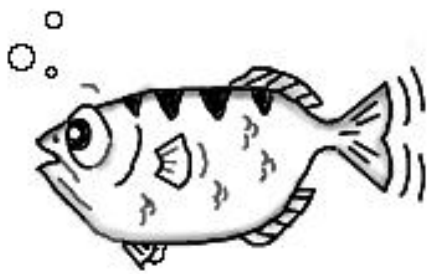
Side Note: gdb



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.

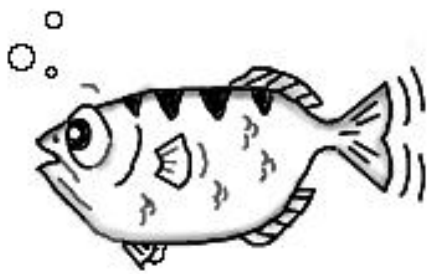
Side Note: gdb



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.

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 the lab machines (via command-line and the IDE `spyder`) and on-line (`onlinegdb.com`).

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`)

Challenge:...

Convert the C++ code to a **Python program**:

```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;
}
```

Python Tutor

Convert the C++ code to a **Python program**:

```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;
}
```

(Write from scratch in `pythonTutor`.)

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`)

Challenge:

Predict what the following pieces of code will do:

```
//Growth example
#include <iostream>
using namespace std;

int main ()
{
  int population = 100;
  cout << "Year\tPopulation\n";
  for (int year = 0; year < 100; year= year+5)
  {
      cout << year << "\t" << population << "\n";
      population = population * 2;

  }
  return 0;
}
```

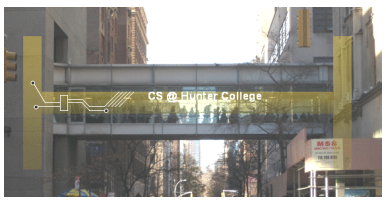
Challenge:

Translate the C++ program into Python:

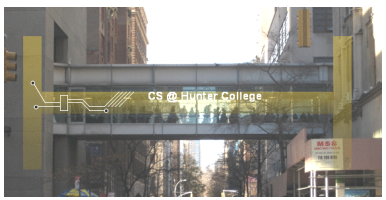
```cpp
//Growth example
#include <iostream>
using namespace std;

int main ()
{
  int population = 100;
  cout << "Year\tPopulation\n";
  for (int year = 0; year < 100; year= year+5)
  {
      cout << year << "\t" << population << "\n";
      population = population * 2;

  }
  return 0;
}
```

Recap: C++

- C++ is a popular programming language that extends C.

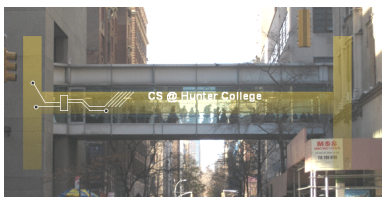


Recap: C++



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

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 Exam: When

- The final exam is **Monday, 18 May, 9am-10:30am**, administered via **Gradescope**.

Final Exam: When

- The final exam is **Monday, 18 May, 9am-10:30am**, administered via **Gradescope**.
- If you have a conflict, the alternative time is:
  Friday, 15 May, 8-9:30am, administered via Gradescope.

Final Exam: When

- The final exam is **Monday, 18 May, 9am-10:30am**, administered via **Gradescope**.
- If you have a conflict, the alternative time is: Friday, 15 May, 8-9:30am, administered via Gradescope.
- You will be added to a different Gradescope course called **CSci 127 Final Exam** for your **version**

Final Exam: When

- The final exam is **Monday, 18 May, 9am-10:30am**, administered via **Gradescope**.
- If you have a conflict, the alternative time is: Friday, 15 May, 8-9:30am, administered via Gradescope.
- You will be added to a different Gradescope course called **CSci 127 Final Exam** for your **version**
- If you have accommodations via the Accessibility Office, please email Prof. Ligorio (tligorio@hunter.cuny.edu) by May 11.

Final Exam: When

- The final exam is **Monday, 18 May, 9am-10:30am**, administered via **Gradescope**.
- If you have a conflict, the alternative time is: Friday, 15 May, 8-9:30am, administered via Gradescope.
- You will be added to a different Gradescope course called **CSci 127 Final Exam** for your **version**
- If you have accommodations via the Accessibility Office, please email Prof. Ligorio (tligorio@hunter.cuny.edu) by May 11.
- **IMPORTANT** let us know your desired exam time and accommodations by answering this survey **by Monday May 11** (Link also provided below video lecture. If you do not answer this survey we will **assume you will take the exam on Monday May 18 at 9am with no accommodations.**)

Final Exam: CR/NC

- Please read instructions for the CR/NC option here:
  (Click on the link if you are reading the pdf or find the clickable link under the video lecture)

Final Exam: CR/NC

- Please read instructions for the CR/NC option here:
  (Click on the link if you are reading the pdf or find the clickable link under the video lecture)
- Students will have up to **June 25**, twenty business days after the University's final grade submission deadline (May 28), to elect CR/NC

Final Exam: CR/NC

- Please read instructions for the CR/NC option here:
  (Click on the link if you are reading the pdf or find the clickable link under the video lecture)
- Students will have up to **June 25**, twenty business days after the University's final grade submission deadline (May 28), to elect CR/NC
- The CR/NC Taskforce is currently in the process of developing a timeline for the communication and implementation that will be rolled out throughout the next 5 weeks.

Final Exam: CR/NC

- Please read instructions for the CR/NC option here:
  (Click on the link if you are reading the pdf or find the clickable link under the video lecture)
- Students will have up to **June 25**, twenty business days after the University's final grade submission deadline (May 28), to elect CR/NC
- The CR/NC Taskforce is currently in the process of developing a timeline for the communication and implementation that will be rolled out throughout the next 5 weeks.
- Students will use CUNYfirst to elect the credit/no credit option. Detailed instructions and communications will be provided across multiple channels as part of the communication and implementation plan.

Final Overview: Format

- Although the exam is remote, we still suggest you prepare 1 piece of **8.5" x 11"** paper.

Final Overview: Format

- Although the exam is remote, we still suggest you prepare 1 piece of **8.5" x 11"** paper.
  - With notes, examples, programs: what will help you on the exam.

Final Overview: Format

- Although the exam is remote, we still suggest you prepare 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 since excellent way to study.

Final Overview: Format

- Although the exam is remote, we still suggest you prepare 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 since excellent way to study.
- The exam format:

Final Overview: Format

- Although the exam is remote, we still suggest you prepare 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 since excellent way to study.
- The exam format:
  - 10 questions, each worth 10 points.

Final Overview: Format

- Although the exam is remote, we still suggest you prepare 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 since excellent way to study.
- The exam format:
  - 10 questions, each worth 10 points.
  - Questions correspond to the course topics, and are variations on the programming assignments, lab exercises, and lecture design challenges.

Final Overview: Format

- Although the exam is remote, we still suggest you prepare 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 since excellent way to study.
- The exam format:
  - 10 questions, each worth 10 points.
  - Questions correspond to the course topics, and are variations on the programming assignments, lab exercises, and lecture design challenges.
  - Style of questions: what does the code do? short answer, write functions, top down design, & write complete programs.

Final Overview: Format

- Although the exam is remote, we still suggest you prepare 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 since excellent way to study.
- The exam format:
  - 10 questions, each worth 10 points.
  - Questions correspond to the course topics, and are variations on the programming assignments, lab exercises, and lecture design challenges.
  - Style of questions: what does the code do? short answer, write functions, top down design, & 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.

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).

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:

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).

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).

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).

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.

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.

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.

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**:

Final Overview: Rules

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

- Your answer uses language constructs that were covered in the course.

Final Overview: Rules

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

- Your answer uses language constructs that were covered in the course.
- Even if your answer is correct, it will get 0 points if the method was not covered in this course.

Final Overview: Rules

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

- Your answer uses language constructs that were covered in the course.
- Even if your answer is correct, it will get 0 points if the method was not covered in this course.
- Your answer is not obviously copy/pasted from a website.

Final Overview: Rules

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

- Your answer uses language constructs that were covered in the course.
- Even if your answer is correct, it will get 0 points if the method was not covered in this course.
- Your answer is not obviously copy/pasted from a website.
- Your answer is not oddly identically to that of another student.

Final Overview: Rules

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

- Your answer uses language constructs that were covered in the course.
- Even if your answer is correct, it will get 0 points if the method was not covered in this course.
- Your answer is not obviously copy/pasted from a website.
- Your answer is not oddly identically to that of another student.

All acts of academic dishonesty will be reported to the Office of Academic and Student Affairs

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.**

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.**

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.**

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)
```

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 whole number and returns the corresponding binary number as a string.**

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 whole number and returns the corresponding binary number as a string.**

```
def num2bin(num):
    ...
    return(bin)
```

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 whole number and returns the corresponding binary number as a string.**

```
def num2bin(num):
    binStr = ""
    while (num > 0):
        #Divide by 2, and add the remainder to the string
        r = num %2
        binString = str(r) + binStr
        num = num / 2
    return(binStr)
```

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 computes the total monthly payment when given the initial loan amount, annual interest rate, number of years of the loan.**

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 computes the total monthly payment when given the initial loan amount, annual interest rate, number of years of the loan.**

```
def computePayment(loan,rate,year):
    ....
    return(payment)
```

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 computes the total monthly payment when given the initial loan amount, annual interest rate, number of years of the loan.**

```
def computePayment(loan,rate,year):
    (Some formula for payment)
    return(payment)
```

Educational Psychology Study

- If you have consented to participate in the **Educational Psychology study**, please fill in the 3-question survey
- Clickable link also below the video.
- Thank you for your participation!!!