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:
  - 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
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
Today’s Topics

- Design Patterns: Searching
- Python Recap
- Machine Language
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- Final Exam: Format
Today’s Topics

- Design Patterns: Searching
- Python Recap
- Machine Language
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- Final Exam: Format
Predict what the code will do:

```python
def search(nums, locate):
    found = False
    i = 0
    while not found and i < len(nums):
        print(nums[i])
        if locate == nums[i]:
            found = True
        else:
            i = i+1
    return found

nums = [1, 4, 10, 6, 5, 42, 9, 8, 12]
if search(nums, 6):
    print('Found it! 6 is in the list!')
else:
    print('Did not find 6 in the list.' )
```
def search(nums, locate):
    found = False
    i = 0
    while not found and i < len(nums):
        print(nums[i])
        if locate == nums[i]:
            found = True
        else:
            i = i+1
    return(found)

nums = [1,4,10,6,5,42,9,8,12]
if search(nums,6):
    print('Found it! 6 is in the list!')
else:
    print('Did not find 6 in the list.')
Design Pattern: Linear Search

Example of linear search.

```python
def search(nums, locate):
    found = False
    i = 0
    while not found and i < len(nums):
        print(nums[i])
        if locate == nums[i]:
            found = True
        else:
            i = i+1
    return(found)
```

```
nums = [1,4,10,6,5,42,9,8,12]
if search(nums,6):
    print('Found it! 6 is in the list!')
else:
    print('Did not find 6 in the list.')
```
Design Pattern: Linear Search

- Example of **linear search**.
- Start at the beginning of the list.

```python
def search(nums, locate):
    found = False
    i = 0
    while not found and i < len(nums):
        print(nums[i])
        if locate == nums[i]:
            found = True
        else:
            i = i + 1
    return (found)

nums = [1,4,10,6,5,42,9,8,12]
if search(nums, 6):
    print('Found it! 6 is in the list!')
else:
    print('Did not find 6 in the list.')```
Design Pattern: Linear Search

- Example of **linear search**.
- Start at the beginning of the list.
- Look at each item, one-by-one.

```python
def search(nums, locate):
    found = False
    i = 0
    while not found and i < len(nums):
        print(nums[i])
        if locate == nums[i]:
            found = True
        else:
            i = i+1
    return(found)
```

```python
nums= [1,4,10,6,5,42,9,8,12]
if search(nums,6):
    print('Found it! 6 is in the list!')
else:
    print('Did not find 6 in the list.')```
Example of **linear search**.

- Start at the beginning of the list.
- Look at each item, one-by-one.
- Stopping, when found, or the end of list is reached.
Today’s Topics

- Design Patterns: Searching
- **Python Recap**
- Machine Language
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- Final Exam: Format
Python & Circuits Review: 10 Weeks in 10 Minutes

A whirlwind tour of the semester, so far...
Week 1: print(), loops, comments, & turtles
Week 1: print(), loops, comments, & turtles

- Introduced comments & print():

  #Name: Thomas Hunter
  #Date: September 1, 2017
  #This program prints: Hello, World!

  print("Hello, World!")

  ← These lines are comments
  ← (for us, not computer to read)
  ← (this one also)
  ← Prints the string "Hello, World!" to the screen
Week 1: print(), loops, comments, & turtles

- Introduced comments & print():

```
#Name: Thomas Hunter
#Date: September 1, 2017
#This program prints: Hello, World!

print("Hello, World!")
```

- As well as definite loops & the turtle package:
A variable is a reserved memory location for storing a value. Different kinds, or types, of values need different amounts of space:

- **int**: integer or whole numbers
- **float**: floating point or real numbers
- **string**: sequence of characters
- **list**: a sequence of items, e.g. `[3, 1, 4, 5, 9]` or `['violet', 'purple', 'indigo']`

More on loops & ranges:
A **variable** is a reserved memory location for storing a value.
Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
  - **int**: integer or whole numbers
  
  ▶ **float**: floating point or real numbers
  
  ▶ **string**: sequence of characters
  
  ▶ **list**: a sequence of items e.g. 
    - [3, 1, 4, 5, 9]
    - ['violet','purple','indigo']
  
  ▶ **class variables**: for complex objects, like turtles.
A variable is a reserved memory location for storing a value.

Different kinds, or types, of values need different amounts of space:

- **int**: integer or whole numbers
- **float**: floating point or real numbers

**class variables**: for complex objects, like turtles.
A variable is a reserved memory location for storing a value.

Different kinds, or types, of values need different amounts of space:

- **int**: integer or whole numbers
- **float**: floating point or real numbers
- **string**: sequence of characters

More on loops & ranges:
Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
  - **int**: integer or whole numbers
  - **float**: floating point or real numbers
  - **string**: sequence of characters
  - **list**: a sequence of items
Week 2: variables, data types, more on loops & range()

- A variable is a reserved memory location for storing a value.
- Different kinds, or types, of values need different amounts of space:
  - `int`: integer or whole numbers
  - `float`: floating point or real numbers
  - `string`: sequence of characters
  - `list`: a sequence of items
    - e.g. `[3, 1, 4, 5, 9]` or `['violet','purple','indigo']`
Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
  - **int**: integer or whole numbers
  - **float**: floating point or real numbers
  - **string**: sequence of characters
  - **list**: a sequence of items
    - e.g. `[3, 1, 4, 5, 9]` or `['violet','purple','indigo']`
  - **class variables**: for complex objects, like turtles.
A **variable** is a reserved memory location for storing a value.

Different kinds, or **types**, of values need different amounts of space:

- **int**: integer or whole numbers
- **float**: floating point or real numbers
- **string**: sequence of characters
- **list**: a sequence of items
- **class variables**: for complex objects, like turtles.

More on loops & ranges:

```python
# Predict what will be printed:
for num in [2, 4, 6, 8, 10]:
    print(num)

sum = 0
for x in range(0, 12, 2):
    print(x)
    sum = sum + x

print(sum)
for c in "ABCD":
    print(c)
```
Week 3: colors, hex, slices, numpy & images

<table>
<thead>
<tr>
<th>Color Name</th>
<th>HEX</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td>#000000</td>
<td><img src="image" alt="Black Color" /></td>
</tr>
<tr>
<td>Navy</td>
<td>#000080</td>
<td><img src="image" alt="Navy Color" /></td>
</tr>
<tr>
<td>DarkBlue</td>
<td>#00008B</td>
<td><img src="image" alt="DarkBlue Color" /></td>
</tr>
<tr>
<td>MediumBlue</td>
<td>#0000CD</td>
<td><img src="image" alt="MediumBlue Color" /></td>
</tr>
<tr>
<td>Blue</td>
<td>#0000FF</td>
<td><img src="image" alt="Blue Color" /></td>
</tr>
</tbody>
</table>

![Hands Diagram](image)
### Week 3: colors, hex, slices, numpy & images

<table>
<thead>
<tr>
<th>Color Name</th>
<th>HEX</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td>#000000</td>
<td><img src="https://via.placeholder.com/150" alt="Black" /></td>
</tr>
<tr>
<td>Navy</td>
<td>#000080</td>
<td><img src="https://via.placeholder.com/150" alt="Navy" /></td>
</tr>
<tr>
<td>DarkBlue</td>
<td>#00008B</td>
<td><img src="https://via.placeholder.com/150" alt="DarkBlue" /></td>
</tr>
<tr>
<td>MediumBlue</td>
<td>#0000CD</td>
<td><img src="https://via.placeholder.com/150" alt="MediumBlue" /></td>
</tr>
<tr>
<td>Blue</td>
<td>#0000FF</td>
<td><img src="https://via.placeholder.com/150" alt="Blue" /></td>
</tr>
</tbody>
</table>

© www.scratchapixel.com
### Week 3: colors, hex, slices, numpy & images

<table>
<thead>
<tr>
<th>Color Name</th>
<th>HEX</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td>#000000</td>
<td>Black</td>
</tr>
<tr>
<td>Navy</td>
<td>#000080</td>
<td>Blue</td>
</tr>
<tr>
<td>DarkBlue</td>
<td>#000888</td>
<td>Blue</td>
</tr>
<tr>
<td>MediumBlue</td>
<td>#0000CD</td>
<td>Blue</td>
</tr>
<tr>
<td>Blue</td>
<td>#0000FF</td>
<td>Blue</td>
</tr>
</tbody>
</table>

```python
>>> a[0, 3:5]
array([[3, 4]])

>>> a[4:, 4:]  
array([[[44, 45],
        [54, 55]]])

>>> a[::, 2]  
array([[2, 12, 22, 32, 42, 52]])

>>> a[2::2, ::2]  
array([[[20, 22, 24],
        [40, 42, 44]]])
```

© www.scratchapixel.com
Week 4: design problem (cropping images) & decisions

First: specify inputs/outputs.
- Input file name
- Output file name
- Upper, lower, left, right ("bounding box")

Next: write pseudocode.
1. Import numpy and pyplot.
2. Ask user for file names and dimensions for cropping.
3. Save input file to an array.
4. Copy the cropped portion to a new array.
5. Save the new array to the output file.

Next: translate to Python.
Week 4: design problem (cropping images) & decisions

- First: specify inputs/outputs. *Input file name, output file name, upper, lower, left, right ("bounding box")*
First: specify inputs/outputs. *Input file name, output file name, upper, lower, left, right ("bounding box")*

Next: write pseudocode.

1. Import numpy and pyplot.
2. Ask user for file names and dimensions for cropping.
3. Save input file to an array.
4. Copy the cropped portion to a new array.
5. Save the new array to the output file.
Week 4: design problem (cropping images) & decisions

- First: specify inputs/outputs. *Input file name, output file name, upper, lower, left, right ("bounding box")*
- Next: write pseudocode.
  1. Import numpy and pyplot.
  2. Ask user for file names and dimensions for cropping.
  3. Save input file to an array.
  4. Copy the cropped portion to a new array.
  5. Save the new array to the output file.
- Next: translate to Python.
Week 4: design problem (cropping images) & decisions

```python
yearBorn = int(input('Enter year born: '))
if yearBorn < 1946:
    print("Greatest Generation")
elif yearBorn <= 1964:
    print("Baby Boomer")
elif yearBorn <= 1984:
    print("Generation X")
elif yearBorn <= 2004:
    print("Millennial")
else:
    print("TBD")

x = int(input('Enter number: '))
if x % 2 == 0:
    print('Even number')
else:
    print('Odd number')
```
Week 5: logical operators, truth tables & logical circuits

```python
origin = "Indian Ocean"
winds = 100
if (winds > 74):
    print("Major storm, called a ", end="")
    if origin == "Indian Ocean" or origin == "South Pacific":
        print("cyclone.")
    elif origin == "North Pacific":
        print("typhoon.")
    else:
        print("hurricane.")

visibility = 0.2
winds = 40
conditions = "blowing snow"
if (winds > 35) and (visibility < 0.25) and \
    (conditions == "blowing snow" or conditions == "heavy snow"):
    print("Blizzard!")
```
Week 5: logical operators, truth tables & logical circuits

<table>
<thead>
<tr>
<th>in1</th>
<th>in2</th>
<th>returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

```
origin = "Indian Ocean"
winds = 100
if (winds > 74):
    print("Major storm, called a ", end="")
    if origin == "Indian Ocean" or origin == "South Pacific":
        print("cyclone."")
    elif origin == "North Pacific":
        print("typhoon."")
    else:
        print("hurricane."")

visibility = 0.2
winds = 40
conditions = "blowing snow"
if (winds > 35) and (visibility < 0.25) and \\
    (conditions == "blowing snow" or conditions == "heavy snow"):
    print("Blizzard!")
```
Week 6: structured data, pandas, & more design

nycHistPop.csv

In Lab 6
import matplotlib.pyplot as plt
import pandas as pd

nycHistPop.csv

In Lab 6
Week 6: structured data, pandas, & more design

import matplotlib.pyplot as plt
import pandas as pd

pop = pd.read_csv('nycHistPop.csv',skiprows=5)

nycHistPop.csv

In Lab 6
Week 6: structured data, pandas, & more design

import matplotlib.pyplot as plt
import pandas as pd

pop = pd.read_csv('nycHistPop.csv',skiprows=5)

pop.plot(x="Year")
plt.show()
import matplotlib.pyplot as plt
import pandas as pd

pop = pd.read_csv('nycHistPop.csv',skiprows=5)

pop.plot(x="Year")
plt.show()
Functions are a way to break code into pieces, that can be easily reused.

```
#Name: your name here
#Date:  October 2017
#This program, uses functions,
#    says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```
Functions are a way to break code into pieces, that can be easily reused.

Many languages require that all code must be organized with functions.

```python
# Name: your name here
# Date: October 2017
# This program uses functions,
#      says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```
Week 7: functions

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`.
Week 7: functions

Functions are a way to break code into pieces, that can be easily reused.

Many languages require that all code must be organized with functions.

The opening function is often called main()

You call or invoke a function by typing its name, followed by any inputs, surrounded by parenthesis:

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
Functions are a way to break code into pieces, that can be easily reused.

Many languages require that all code must be organized with functions.

The opening function is often called `main()`

You call or invoke a function by typing its name, followed by any inputs, surrounded by parenthesis:

Example: `print("Hello", "World")`

```python
# Name: your name here
# Date: October 2017
# This program, uses functions, 
# says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```
Week 7: functions

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`.
- You call or invoke a function by typing its name, followed by any inputs, surrounded by parenthesis:
  Example: `print("Hello", "World")`
- Can write, or define your own functions,

```python
# This program, uses functions, says hello to the world!
def main():
    print("Hello, World!")
if __name__ == "__main__":
    main()
```
Week 7: functions

Functions are a way to break code into pieces, that can be easily reused.

Many languages require that all code must be organized with functions.

The opening function is often called main()

You call or invoke a function by typing its name, followed by any inputs, surrounded by parenthesis:
Example: print("Hello", "World")

Can write, or define your own functions, which are stored, until invoked or called.
Week 8: function parameters, github

- Functions can have **input parameters**.

```python
def totalWithTax(food, tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip: '))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

dinner = float(input('Enter dinner total: '))
dTip = float(input('Enter dinner tip: '))
dTotal = totalWithTax(dinner, dTip)
print('Dinner total is', dTotal)
```
Functions can have **input parameters**.

Surrounded by parenthesis, both in the function definition, and in the function call (invocation).

```python
def totalWithTax(food, tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip: '))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

dinner= float(input('Enter dinner total: '))
dTip = float(input('Enter dinner tip: '))
dTotal = totalWithTax(dinner, dTip)
print('Dinner total is', dTotal)
```
Functions can have **input parameters**.

Surrounded by parenthesis, both in the function definition, and in the function call (invocation).

The “placeholders” in the function definition: **formal parameters**.
Week 8: function parameters, github

- Functions can have **input parameters**.
- Surrounded by parenthesis, both in the function definition, and in the function call (invocation).
- The “placeholders” in the function definition: **formal parameters**.
- The ones in the function call: **actual parameters**.

```python
def totalWithTax(food, tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
ltip = float(input('Enter lunch tip: '))
lTotal = totalWithTax(lunch, ltip)
print('Lunch total is', lTotal)

dinner = float(input('Enter dinner total: '))
dtip = float(input('Enter dinner tip: '))
dTotal = totalWithTax(dinner, dtip)
print('Dinner total is', dTotal)
```
Functions can have **input parameters**.

Surrounded by parenthesis, both in the function definition, and in the function call (invocation).

The “placeholders” in the function definition: **formal parameters**.

The ones in the function call: **actual parameters**

Functions can also **return values** to where it was called.
Week 8: function parameters, github

- Functions can have **input parameters**.
- Surrounded by parenthesis, both in the function definition, and in the function call (invocation).
- The “placeholders” in the function definition: **formal parameters**.
- The ones in the function call: **actual parameters**.
- Functions can also **return values** to where it was called.

```python
def totalWithTax(food, tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip: '))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

dinner = float(input('Enter dinner total: '))
dTip = float(input('Enter dinner tip: '))
dTotal = totalWithTax(dinner, dTip)
priint('Dinner total is', dTotal)
```
def main():
    dataF = getData()
    latColName, lonColName = getColumnNames()
    lat, lon = getLocale()
    cityMap = folium.Map(location=[lat, lon], tiles='cartodbpositron', zoom_start=11)
    dotAllPoints(cityMap, dataF, latColName, lonColName)
    markAndFindClosest(cityMap, dataF, latColName, lonColName, lat, lon)
    writeMap(cityMap)
Week 10: more on loops, max design pattern, random()

- Indefinite (while) loops allow you to repeat a block of code as long as a condition holds.

```python
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))

print('The distance entered is', dist)
```

```python
import turtle
import random

trey = turtle.Turtle()
trey.speed(10)

for i in range(100):
    trey.forward(10)
    a = random.randrange(0,360,90)
    trey.right(a)
```
Week 10: more on loops, max design pattern, random()

Indefinite (while) loops allow you to repeat a block of code as long as a condition holds.

- Very useful for checking user input for correctness.

```python
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))
print('The distance entered is', dist)
```

Python's built-in random package has useful methods for generating random whole numbers and real numbers.

To use, must include:
```
import random
```

The max design pattern provides a template for finding maximum value from a list.

```python
import turtle
import random

trey = turtle.Turtle()
trey.speed(10)

for i in range(100):
    trey.forward(10)
    a = random.randrange(0,360,90)
    trey.right(a)
```
Indefinite (while) loops allow you to repeat a block of code as long as a condition holds.

- Very useful for checking user input for correctness.
- Python's built-in random package has useful methods for generating random whole numbers and real numbers.
Indefinite (while) loops allow you to repeat a block of code as long as a condition holds.

- Very useful for checking user input for correctness.

- Python’s built-in random package has useful methods for generating random whole numbers and real numbers.

- To use, must include:
  ```python
  import random
  ```
Indefinite (while) loops allow you to repeat a block of code as long as a condition holds.

Very useful for checking user input for correctness.

Python’s built-in random package has useful methods for generating random whole numbers and real numbers.

To use, must include:

```
import random
```

The max design pattern provides a template for finding maximum value from a list.
Input/Output (I/O): `input()` and `print()`;
pandas for CSV files

Types:
- Primitive: int, float, bool, string;
- Container: lists (but not dictionaries/hashe
  or tuples)

Objects: turtles (used but did not design our own)

Loops: definite & indefinite

Conditionals: if-elif-else

Logical Expressions & Circuits

Functions: parameters & returns

Packages:
- Built-in: turtle, math, random
- Popular: numpy, matplotlib, pandas, folium
Today’s Topics

- Design Patterns: Searching
- Python Recap
- **Machine Language**
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- Final Exam: Format
Can view programming languages on a continuum.
Can view programming languages on a continuum.

Those that directly access machine instructions & memory and have little abstraction are **low-level languages**.
Low-Level vs. High-Level Languages

- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).
Low-Level vs. High-Level Languages

- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).
- Those that have strong abstraction (allow programming paradigms independent of the machine details, such as complex variables, functions and looping that do not translate directly into machine code) are called **high-level languages**.
Low-Level vs. High-Level Languages

- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).
- Those that have strong abstraction (allow programming paradigms independent of the machine details, such as complex variables, functions and looping that do not translate directly into machine code) are called **high-level languages**.
- Some languages, like C, are in between— allowing both low level access and high level data structures.
Machine Language

(Ruth Gordon & Ester Gerston programming the ENIAC, UPenn)
Machine Language

(img)

(wiki)
Machine Language

We will be writing programs in a simplified machine language, WeMIPS.

(wiki)
We will be writing programs in a simplified machine language, WeMIPS.

It is based on a reduced instruction set computer (RISC) design, originally developed by the MIPS Computer Systems.
We will be writing programs in a simplified machine language, WeMIPS.

It is based on a reduced instruction set computer (RISC) design, originally developed by the MIPS Computer Systems.

Due to its small set of commands, processors can be designed to run those commands very efficiently.
Machine Language

- We will be writing programs in a simplified machine language, WeMIPS.
- It is based on a reduced instruction set computer (RISC) design, originally developed by the MIPS Computer Systems.
- Due to its small set of commands, processors can be designed to run those commands very efficiently.
- More in future architecture classes....
“Hello World!” in Simplified Machine Language

```assembly
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)
ADDI $s0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall   # print to the log
```

(WeMIPS)
WeMIPS

(Demo with WeMIPS)
MIPS Commands

- **Registers**: locations for storing information that can be quickly accessed.
MIPS Commands

- **Registers**: locations for storing information that can be quickly accessed. Names start with ‘$’: $s0, $s1, $t0, $t1,...
**MIPS Commands**

- **Registers**: locations for storing information that can be quickly accessed. Names start with ‘$’: $s0, $s1, $t0, $t1,...

- **R Instructions**: Commands that use data in the registers:
**MIPS Commands**

- **Registers**: locations for storing information that can be quickly accessed. Names start with `'$'`: `$s0`, `$s1`, `$t0`, `$t1`,...

- **R Instructions**: Commands that use data in the registers: `add $s1, $s2, $s3`
MIPS Commands

- **Registers**: locations for storing information that can be quickly accessed. Names start with ‘$’: $s0, $s1, $t0, $t1,...

- **R Instructions**: Commands that use data in the registers:
  - `add $s1, $s2, $s3` (Basic form: `OP rd, rs, rt`)

- **I Instructions**: instructions that also use intermediate values.
MIPS Commands

- **Registers**: locations for storing information that can be quickly accessed. Names start with ‘$’: $s0, $s1, $t0, $t1,...

- **R Instructions**: Commands that use data in the registers: 
  \[
  \text{add } \quad \text{OP rd, rs, rt}
  \]
  (Basic form: OP rd, rs, rt)

- **I Instructions**: instructions that also use intermediate values.
  \[
  \text{addi } \quad \text{OP rd, rs, imm}
  \]
  \[
  \text{addi } \quad \text{OP rd, rs, imm}
  \]
MIPS Commands

- **Registers**: locations for storing information that can be quickly accessed. Names start with ‘$’: $s0, $s1, $t0, $t1,...

- **R Instructions**: Commands that use data in the registers:
  \[ \text{add} \; $s1, \; $s2, \; $s3 \quad \text{(Basic form: OP rd, rs, rt)} \]

- **I Instructions**: instructions that also use intermediate values.
  \[ \text{addi} \; $s1, \; $s2, \; 100 \quad \text{(Basic form: OP rd, rs, imm)} \]

- **J Instructions**: instructions that jump to another memory location.
MIPS Commands

- **Registers:** locations for storing information that can be quickly accessed. Names start with ‘$’: $s0, $s1, $t0, $t1,…

- **R Instructions:** Commands that use data in the registers:
  
  ```
  add $s1, $s2, $s3  
  ```
  
  (Basic form: OP rd, rs, rt)

- **I Instructions:** instructions that also use intermediate values.
  
  ```
  addi $s1, $s2, 100  
  ```
  
  (Basic form: OP rd, rs, imm)

- **J Instructions:** instructions that jump to another memory location.
  
  ```
  j done  
  ```
MIPS Commands

- **Registers**: locations for storing information that can be quickly accessed. Names start with ‘$’: $s0, $s1, $t0, $t1,...

- **R Instructions**: Commands that use data in the registers:
  
  \[
  \text{add } \text{ } s1, s2, s3 \quad \text{(Basic form: OP rd, rs, rt)}
  \]

- **I Instructions**: instructions that also use intermediate values.
  
  \[
  \text{addi } s1, s2, 100 \quad \text{(Basic form: OP rd, rs, imm)}
  \]

- **J Instructions**: instructions that jump to another memory location.
  
  \[
  \text{j } \text{done} \quad \text{(Basic form: OP label)}
  \]
In Pairs or Triples:

Write a program that prints out the alphabet: a b c d ... x y z
WeMIPS

(Demo with WeMIPS)
Today’s Topics

- Design Patterns: Searching
- Python Recap
- Machine Language
- **Machine Language: Jumps & Loops**
- Binary & Hex Arithmetic
- Final Exam: Format
Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.

Different kinds of jumps:

- Unconditional: `j Done` will jump to the address with label `Done`.
- Branch if Equal: `beq $s0 $s1 DoAgain` will jump to the address with label `DoAgain` if the registers `$s0` and `$s1` contain the same value.

See reading for more variations.
Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.
- Can indicate locations by writing **labels** at the beginning of a line.
Loops & Jumps in Machine Language

• Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.

• Can indicate locations by writing `labels` at the beginning of a line.

• Then give a command to jump to that location.
Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.
- Can indicate locations by writing `labels` at the beginning of a line.
- Then give a command to jump to that location.
- Different kinds of jumps:

- **Unconditional:** `j Done` will jump to the address with label `Done`.
- **Branch if Equal:** `beq $s0 $s1 DoAgain` will jump to the address with label `DoAgain` if the registers $s0 and $s1 contain the same value.
- See reading for more variations.
Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.
- Can indicate locations by writing **labels** at the beginning of a line.
- Then give a command to jump to that location.
- Different kinds of jumps:
  - **Unconditional**: `j Done` will jump to the address with label `Done`.
- See reading for more variations.
Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.
- Can indicate locations by writing **labels** at the beginning of a line.
- Then give a command to jump to that location.
- Different kinds of jumps:
  - **Unconditional:** `j Done` will jump to the address with label `Done`.
  - **Branch if Equal:** `beq $s0 $s1 DoAgain` will jump to the address with label `DoAgain` if the registers `$s0` and `$s1` contain the same value.

▶ See reading for more variations.
Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.
- Can indicate locations by writing `labels` at the beginning of a line.
- Then give a command to jump to that location.
- Different kinds of jumps:
  - **Unconditional**: `j Done` will jump to the address with label `Done`.
  - **Branch if Equal**: `beq $s0 $s1 DoAgain` will jump to the address with label `DoAgain` if the registers $s0 and $s1 contain the same value.
  - See reading for more variations.
Jump Demo

(Demo with WeMIPS)
Today’s Topics

- Design Patterns: Searching
- Python Recap
- Machine Language
- Machine Language: Jumps & Loops
- **Binary & Hex Arithmetic**
- Final Exam: Format
From hexadecimal to decimal:

- Convert first digit to decimal and multiple by 16.
- Convert second digit to decimal and add to total.
Hexadecimal to Decimal: Converting Between Bases

(from i-programmer.info)

- From hexadecimal to decimal:
  - Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.

Example: what is 2A as a decimal number?
- 2 in decimal is 2.
- 2*16 is 32.
- A in decimal digits is 10.
- 32 + 10 is 42.
- Answer is 42.

Example: what is 99 as a decimal number?
- 9 in decimal is 9.
- 9*16 is 144.
- 9 in decimal digits is 9.
- 144 + 9 is 153.
- Answer is 153.
Hexadecimal to Decimal: Converting Between Bases

(from i-programmer.info)

- From hexadecimal to decimal:
  - Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?

  2 in decimal is 2.
  2*16 is 32.
  A in decimal digits is 10.
  32 + 10 is 42.
  Answer is 42.

  9 in decimal is 9.
  9*16 is 144.
  9 in decimal digits is 9.
  144 + 9 is 153.
  Answer is 153.
From hexadecimal to decimal:

- Convert first digit to decimal and multiple by 16.
- Convert second digit to decimal and add to total.
- Example: what is 2A as a decimal number?
  
  2 in decimal is 2.

\[ 2 \times 16 = 32 \]

\[ 32 + 10 = 42 \]

Answer is 42.

- Example: what is 99 as a decimal number?

\[ 9 \times 16 = 144 \]

\[ 144 + 9 = 153 \]

Answer is 153.
Hexadecimal to Decimal: Converting Between Bases

(from i-programmer.info)

- From hexadecimal to decimal:
  - Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?
    - 2 in decimal is 2. 2*16 is 32.
  - Example: what is 99 as a decimal number?
    - 9 in decimal is 9.
    - 9*16 is 144.
    - 9 in decimal digits is 9
    - 144 + 9 is 153.
    - Answer is 153.
Hexadecimal to Decimal: Converting Between Bases

From hexadecimal to decimal:
- Convert first digit to decimal and multiple by 16.
- Convert second digit to decimal and add to total.
- Example: what is 2A as a decimal number?
  - 2 in decimal is 2. 2*16 is 32.
  - A in decimal digits is 10.
  - Answer is 42.

- Example: what is 99 as a decimal number?
  - 9 in decimal is 9.
  - 9*16 is 144.
  - 9 in decimal digits is 9
  - 144 + 9 is 153.
  - Answer is 153.
Hexadecimal to Decimal: Converting Between Bases

(from i-programmer.info)

- From hexadecimal to decimal:
  - Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?
    - 2 in decimal is 2. 2*16 is 32.
    - A in decimal digits is 10.
    - 32 + 10 is 42.
Hexadecimal to Decimal: Converting Between Bases

(from i-programmer.info)

- From hexadecimal to decimal:
  - Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?
    2 in decimal is 2. 2*16 is 32.
    A in decimal digits is 10.
    32 + 10 is 42.
    Answer is 42.
  - Example: what is 99 as a decimal number?
Hexadecimal to Decimal: Converting Between Bases

(from i-programmer.info)

- From hexadecimal to decimal:
  - Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?
    2 in decimal is 2. 2*16 is 32.
    A in decimal digits is 10.
    32 + 10 is 42.
    Answer is 42.
  - Example: what is 99 as a decimal number?
    9 in decimal is 9.
Hexadecimal to Decimal: Converting Between Bases

(from i-programmer.info)

- From hexadecimal to decimal:
  - Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?
    2 in decimal is 2. 2*16 is 32.
    A in decimal digits is 10.
    32 + 10 is 42.
    Answer is 42.
  - Example: what is 99 as a decimal number?
    9 in decimal is 9. 9*16 is 144.
Hexadecimal to Decimal: Converting Between Bases

(from i-programmer.info)

- From hexadecimal to decimal:
  - Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?
    2 in decimal is 2. 2*16 is 32.
    A in decimal digits is 10.
    32 + 10 is 42.
    Answer is 42.
  - Example: what is 99 as a decimal number?
    9 in decimal is 9. 9*16 is 144.
    9 in decimal digits is 9
Hexadecimal to Decimal: Converting Between Bases

- From hexadecimal to decimal:
  - Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?
    - 2 in decimal is 2. 2*16 is 32.
    - A in decimal digits is 10.
    - 32 + 10 is 42.
    - Answer is 42.
  - Example: what is 99 as a decimal number?
    - 9 in decimal is 9. 9*16 is 144.
    - 9 in decimal digits is 9
    - 144 + 9 is 153.
Hexadecimal to Decimal: Converting Between Bases

(from i-programmer.info)

- From hexadecimal to decimal:
  - Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?
    2 in decimal is 2. 2*16 is 32.
    A in decimal digits is 10.
    32 + 10 is 42.
    Answer is 42.
  - Example: what is 99 as a decimal number?
    9 in decimal is 9. 9*16 is 144.
    9 in decimal digits is 9
    144 + 9 is 153.
    Answer is 153.
Decimal to Binary: Converting Between Bases

- From decimal to binary:
  - Divide by 128 \( (= 2^7) \). Quotient is the first digit.
  - Divide remainder by 64 \( (= 2^6) \). Quotient is the next digit.
  - Divide remainder by 32 \( (= 2^5) \). Quotient is the next digit.
  - Divide remainder by 16 \( (= 2^4) \). Quotient is the next digit.
  - Divide remainder by 8 \( (= 2^3) \). Quotient is the next digit.
  - Divide remainder by 4 \( (= 2^2) \). Quotient is the next digit.
  - Divide remainder by 2 \( (= 2^1) \). Quotient is the next digit.
  - The last remainder is the last digit.

Example: what is 130 in binary notation?

130/128 is 1 rem 2.
First digit is 1:
10 ...
2/64 is 0 rem 2.
Next digit is 0:
100 ...
2/32 is 0 rem 2.
Next digit is 0:
1000 ...
2/16 is 0 rem 2.
Next digit is 0:
10000 ...
2/8 is 0 rem 2.
Next digit is 0:
100000 ...
2/4 is 0 remainder 2.
Next digit is 0:
1000000 ...
2/2 is 1 rem 0.
Next digit is 1:
1000001 ...
Adding the last remainder:
10000010
From decimal to binary:

- Divide by 128 ($= 2^7$). Quotient is the first digit.
- Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
From decimal to binary:

- Divide by $128 = 2^7$. Quotient is the first digit.
- Divide remainder by $64 = 2^6$. Quotient is the next digit.
- Divide remainder by $32 = 2^5$. Quotient is the next digit.
- Divide remainder by $16 = 2^4$. Quotient is the next digit.
- Divide remainder by $8 = 2^3$. Quotient is the next digit.
- Divide remainder by $4 = 2^2$. Quotient is the next digit.
- The last remainder is the last digit.

Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1:

$\ldots$ 1

2/64 is 0 rem 2. Next digit is 0:

$\ldots$ 10

2/32 is 0 rem 2. Next digit is 0:

$\ldots$ 100

2/16 is 0 rem 2. Next digit is 0:

$\ldots$ 1000

2/8 is 0 rem 2. Next digit is 0:

$\ldots$ 10000

2/4 is 0 remainder 2. Next digit is 0:

$\ldots$ 100000

2/2 is 1 rem 0. Next digit is 1:

$\ldots$ 1000001

Adding the last remainder:

$\ldots$ 10000010
Decimal to Binary: Converting Between Bases

From decimal to binary:

- Divide by 128 ($= 2^7$). Quotient is the first digit.
- Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- The last remainder is the last digit.

Example: what is 130 in binary notation?

130/128 is 1 rem 2.
First digit is 1: 

2/64 is 0 rem 2.
Next digit is 0: 

2/32 is 0 rem 2.
Next digit is 0: 

2/16 is 0 rem 2.
Next digit is 0: 

2/8 is 0 rem 2.
Next digit is 0: 

2/4 is 0 rem 2.
Next digit is 0: 

2/2 is 1 rem 0.
Next digit is 1: 

Adding the last remainder:

10000010
Decimal to Binary: Converting Between Bases

From decimal to binary:

▶ Divide by 128 (\(= 2^7\)). Quotient is the first digit.
▶ Divide remainder by 64 (\(= 2^6\)). Quotient is the next digit.
▶ Divide remainder by 32 (\(= 2^5\)). Quotient is the next digit.
▶ Divide remainder by 16 (\(= 2^4\)). Quotient is the next digit.
▶ Divide remainder by 8 (\(= 2^3\)). Quotient is the next digit.
▶ The last remainder is the last digit.

Example: what is 130 in binary notation?

\[
130/128 \text{ is } 1 \text{ rem } 2.
\]
First digit is 1:

\[
10
\]

\[
2/64 \text{ is } 0 \text{ rem } 2.
\]
Next digit is 0:

\[
100
\]

\[
2/32 \text{ is } 0 \text{ rem } 2.
\]
Next digit is 0:

\[
1000
\]

\[
2/16 \text{ is } 0 \text{ rem } 2.
\]
Next digit is 0:

\[
10000
\]

\[
2/8 \text{ is } 0 \text{ rem } 2.
\]
Next digit is 0:

\[
100000
\]

\[
2/4 \text{ is } 0 \text{ rem } 2.
\]
Next digit is 0:

\[
1000001
\]

Adding the last remainder:

\[
10000010
\]
From decimal to binary:

- Divide by 128 ($= 2^7$). Quotient is the first digit.
- Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- The last remainder is the last digit.

Example: what is 130 in binary notation?

130/128 is 1 rem 2.
First digit is 1: 1 ...
2/64 is 0 rem 2.
Next digit is 0: 10 ...
2/32 is 0 rem 2.
Next digit is 0: 100 ...
2/16 is 0 rem 2.
Next digit is 0: 1000 ...
2/8 is 0 rem 2.
Next digit is 0: 10000 ...
2/4 is 0 remainder 2.
Next digit is 0: 100000 ...
2/2 is 1 rem 0.
Next digit is 1: 1000001 ...
Adding the last remainder: 10000010
From decimal to binary:

- Divide by 128 ($= 2^7$). Quotient is the first digit.
- Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- The last remainder is the last digit.

Example: what is 130 in binary notation?

130/128 is 1 rem 2.
First digit is 1:

2/64 is 0 rem 2.
Next digit is 0:

2/32 is 0 rem 2.
Next digit is 0:

2/16 is 0 rem 2.
Next digit is 0:

2/8 is 0 rem 2.
Next digit is 0:

2/4 is 0 remainder 2.
Next digit is 0:

2/2 is 1 rem 0.
Next digit is 1:

Adding the last remainder:

10000010
Decimal to Binary: Converting Between Bases

From decimal to binary:

▶ Divide by 128 (= $2^7$). Quotient is the first digit.
▶ Divide remainder by 64 (= $2^6$). Quotient is the next digit.
▶ Divide remainder by 32 (= $2^5$). Quotient is the next digit.
▶ Divide remainder by 16 (= $2^4$). Quotient is the next digit.
▶ Divide remainder by 8 (= $2^3$). Quotient is the next digit.
▶ Divide remainder by 4 (= $2^2$). Quotient is the next digit.
▶ Divide remainder by 2 (= $2^1$). Quotient is the next digit.
▶ The last remainder is the last digit.
Decimal to Binary: Converting Between Bases

From decimal to binary:

- Divide by 128 ($= 2^7$). Quotient is the first digit.
- Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?
Decimal to Binary: Converting Between Bases

From decimal to binary:

- Divide by 128 (\(= 2^7\)). Quotient is the first digit.
- Divide remainder by 64 (\(= 2^6\)). Quotient is the next digit.
- Divide remainder by 32 (\(= 2^5\)). Quotient is the next digit.
- Divide remainder by 16 (\(= 2^4\)). Quotient is the next digit.
- Divide remainder by 8 (\(= 2^3\)). Quotient is the next digit.
- Divide remainder by 4 (\(= 2^2\)). Quotient is the next digit.
- Divide remainder by 2 (\(= 2^1\)). Quotient is the next digit.
- The last remainder is the last digit.

Example: what is 130 in binary notation?

130/128 is 1 rem 2.
Decimal to Binary: Converting Between Bases

From decimal to binary:

- Divide by 128 (\(= 2^7\)). Quotient is the first digit.
- Divide remainder by 64 (\(= 2^6\)). Quotient is the next digit.
- Divide remainder by 32 (\(= 2^5\)). Quotient is the next digit.
- Divide remainder by 16 (\(= 2^4\)). Quotient is the next digit.
- Divide remainder by 8 (\(= 2^3\)). Quotient is the next digit.
- Divide remainder by 4 (\(= 2^2\)). Quotient is the next digit.
- Divide remainder by 2 (\(= 2^1\)). Quotient is the next digit.
- The last remainder is the last digit.

Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1:
Decimal to Binary: Converting Between Bases

From decimal to binary:

- Divide by 128 \( (= 2^7) \). Quotient is the first digit.
- Divide remainder by 64 \( (= 2^6) \). Quotient is the next digit.
- Divide remainder by 32 \( (= 2^5) \). Quotient is the next digit.
- Divide remainder by 16 \( (= 2^4) \). Quotient is the next digit.
- Divide remainder by 8 \( (= 2^3) \). Quotient is the next digit.
- Divide remainder by 4 \( (= 2^2) \). Quotient is the next digit.
- Divide remainder by 2 \( (= 2^1) \). Quotient is the next digit.
- The last remainder is the last digit.

Example: what is 130 in binary notation?

\[
\begin{align*}
130/128 & \text{ is } 1 \text{ rem } 2. \text{ First digit is } 1: \quad 1... \\
2/64 & \text{ is } 0 \text{ rem } 2.
\end{align*}
\]
Decimal to Binary: Converting Between Bases

From decimal to binary:

- Divide by 128 (\(= 2^7\)). Quotient is the first digit.
- Divide remainder by 64 (\(= 2^6\)). Quotient is the next digit.
- Divide remainder by 32 (\(= 2^5\)). Quotient is the next digit.
- Divide remainder by 16 (\(= 2^4\)). Quotient is the next digit.
- Divide remainder by 8 (\(= 2^3\)). Quotient is the next digit.
- Divide remainder by 4 (\(= 2^2\)). Quotient is the next digit.
- Divide remainder by 2 (\(= 2^1\)). Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?
  - \(130/128 = 1 \text{ rem } 2\). First digit is 1: 1...
  - \(2/64 = 0 \text{ rem } 2\). Next digit is 0:
Decimal to Binary: Converting Between Bases

- From decimal to binary:
  - Divide by 128 ( = $2^7$). Quotient is the first digit.
  - Divide remainder by 64 ( = $2^6$). Quotient is the next digit.
  - Divide remainder by 32 ( = $2^5$). Quotient is the next digit.
  - Divide remainder by 16 ( = $2^4$). Quotient is the next digit.
  - Divide remainder by 8 ( = $2^3$). Quotient is the next digit.
  - Divide remainder by 4 ( = $2^2$). Quotient is the next digit.
  - Divide remainder by 2 ( = $2^1$). Quotient is the next digit.
  - The last remainder is the last digit.
- Example: what is 130 in binary notation?
  - 130/128 is 1 rem 2. First digit is 1: 1...
  - 2/64 is 0 rem 2. Next digit is 0: 10...
Decimal to Binary: Converting Between Bases

From decimal to binary:

- Divide by 128 (\(= 2^7\)). Quotient is the first digit.
- Divide remainder by 64 (\(= 2^6\)). Quotient is the next digit.
- Divide remainder by 32 (\(= 2^5\)). Quotient is the next digit.
- Divide remainder by 16 (\(= 2^4\)). Quotient is the next digit.
- Divide remainder by 8 (\(= 2^3\)). Quotient is the next digit.
- Divide remainder by 4 (\(= 2^2\)). Quotient is the next digit.
- Divide remainder by 2 (\(= 2^1\)). Quotient is the next digit.
- The last remainder is the last digit.

Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...
2/64 is 0 rem 2. Next digit is 0: 10...
2/32 is 0 rem 2.
Decimal to Binary: Converting Between Bases

- From decimal to binary:
  - Divide by $128 = 2^7$. Quotient is the first digit.
  - Divide remainder by $64 = 2^6$. Quotient is the next digit.
  - Divide remainder by $32 = 2^5$. Quotient is the next digit.
  - Divide remainder by $16 = 2^4$. Quotient is the next digit.
  - Divide remainder by $8 = 2^3$. Quotient is the next digit.
  - Divide remainder by $4 = 2^2$. Quotient is the next digit.
  - Divide remainder by $2 = 2^1$. Quotient is the next digit.
  - The last remainder is the last digit.

- Example: what is 130 in binary notation?
  - $130/128$ is 1 rem 2. First digit is 1: 1...
  - $2/64$ is 0 rem 2. Next digit is 0: 10...
  - $2/32$ is 0 rem 2. Next digit is 0:
Decimal to Binary: Converting Between Bases

From decimal to binary:

▶ Divide by 128 (\(= 2^7\)). Quotient is the first digit.
▶ Divide remainder by 64 (\(= 2^6\)). Quotient is the next digit.
▶ Divide remainder by 32 (\(= 2^5\)). Quotient is the next digit.
▶ Divide remainder by 16 (\(= 2^4\)). Quotient is the next digit.
▶ Divide remainder by 8 (\(= 2^3\)). Quotient is the next digit.
▶ Divide remainder by 4 (\(= 2^2\)). Quotient is the next digit.
▶ Divide remainder by 2 (\(= 2^1\)). Quotient is the next digit.
▶ The last remainder is the last digit.
▶ Example: what is 130 in binary notation?

\[
\begin{align*}
130/128 & \text{ is 1 rem 2. First digit is 1: } 1\ldots \\
2/64 & \text{ is 0 rem 2. Next digit is 0: } 10\ldots \\
2/32 & \text{ is 0 rem 2. Next digit is 0: } 100\ldots 
\end{align*}
\]
Decimal to Binary: Converting Between Bases

- From decimal to binary:
  - Divide by 128 \( (= 2^7) \). Quotient is the first digit.
  - Divide remainder by 64 \( (= 2^6) \). Quotient is the next digit.
  - Divide remainder by 32 \( (= 2^5) \). Quotient is the next digit.
  - Divide remainder by 16 \( (= 2^4) \). Quotient is the next digit.
  - Divide remainder by 8 \( (= 2^3) \). Quotient is the next digit.
  - Divide remainder by 4 \( (= 2^2) \). Quotient is the next digit.
  - Divide remainder by 2 \( (= 2^1) \). Quotient is the next digit.
  - The last remainder is the last digit.
- Example: what is 130 in binary notation?
  - \( \frac{130}{128} \) is 1 rem 2. First digit is 1: 1...
  - \( \frac{2}{64} \) is 0 rem 2. Next digit is 0: 10...
  - \( \frac{2}{32} \) is 0 rem 2. Next digit is 0: 100...
  - \( \frac{2}{16} \) is 0 rem 2.
From decimal to binary:

- Divide by 128 ($= 2^7$). Quotient is the first digit.
- Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- The last remainder is the last digit.

Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...
2/64 is 0 rem 2. Next digit is 0: 10...
2/32 is 0 rem 2. Next digit is 0: 100...
2/16 is 0 rem 2. Next digit is 0:
Decimal to Binary: Converting Between Bases

From decimal to binary:

- Divide by 128 (\(= 2^7\)). Quotient is the first digit.
- Divide remainder by 64 (\(= 2^6\)). Quotient is the next digit.
- Divide remainder by 32 (\(= 2^5\)). Quotient is the next digit.
- Divide remainder by 16 (\(= 2^4\)). Quotient is the next digit.
- Divide remainder by 8 (\(= 2^3\)). Quotient is the next digit.
- Divide remainder by 4 (\(= 2^2\)). Quotient is the next digit.
- Divide remainder by 2 (\(= 2^1\)). Quotient is the next digit.
- The last remainder is the last digit.

**Example:** what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...
2/64 is 0 rem 2. Next digit is 0: 10...
2/32 is 0 rem 2. Next digit is 0: 100...
2/16 is 0 rem 2. Next digit is 0: 1000...
Decimal to Binary: Converting Between Bases

- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
  - Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
  - Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
  - The last remainder is the last digit.
- Example: what is 130 in binary notation?
  
  
  130/128 is 1 rem 2. First digit is 1: 1...
  2/64 is 0 rem 2. Next digit is 0: 10...
  2/32 is 0 rem 2. Next digit is 0: 100...
  2/16 is 0 rem 2. Next digit is 0: 1000...
  2/8 is 0 rem 2.
Decimal to Binary: Converting Between Bases

- From decimal to binary:
  - Divide by 128 (\(= 2^7\)). Quotient is the first digit.
  - Divide remainder by 64 (\(= 2^6\)). Quotient is the next digit.
  - Divide remainder by 32 (\(= 2^5\)). Quotient is the next digit.
  - Divide remainder by 16 (\(= 2^4\)). Quotient is the next digit.
  - Divide remainder by 8 (\(= 2^3\)). Quotient is the next digit.
  - Divide remainder by 4 (\(= 2^2\)). Quotient is the next digit.
  - Divide remainder by 2 (\(= 2^1\)). Quotient is the next digit.
  - The last remainder is the last digit.
- Example: what is 130 in binary notation?
  130/128 is 1 rem 2. First digit is 1: \[1\ldots\]
  2/64 is 0 rem 2. Next digit is 0: \[10\ldots\]
  2/32 is 0 rem 2. Next digit is 0: \[100\ldots\]
  2/16 is 0 rem 2. Next digit is 0: \[1000\ldots\]
  2/8 is 0 rem 2. Next digit is 0:
Decimal to Binary: Converting Between Bases

- From decimal to binary:
  - Divide by 128 (\(= 2^7\)). Quotient is the first digit.
  - Divide remainder by 64 (\(= 2^6\)). Quotient is the next digit.
  - Divide remainder by 32 (\(= 2^5\)). Quotient is the next digit.
  - Divide remainder by 16 (\(= 2^4\)). Quotient is the next digit.
  - Divide remainder by 8 (\(= 2^3\)). Quotient is the next digit.
  - Divide remainder by 4 (\(= 2^2\)). Quotient is the next digit.
  - Divide remainder by 2 (\(= 2^1\)). Quotient is the next digit.
  - The last remainder is the last digit.

- Example: what is 130 in binary notation?
  
  130/128 is 1 rem 2. First digit is 1: 1...
  
  2/64 is 0 rem 2. Next digit is 0: 10...
  
  2/32 is 0 rem 2. Next digit is 0: 100...
  
  2/16 is 0 rem 2. Next digit is 0: 1000...
  
  2/8 is 0 rem 2. Next digit is 0: 10000...
Decimal to Binary: Converting Between Bases

From decimal to binary:

- Divide by 128 (\(= 2^7\)). Quotient is the first digit.
- Divide remainder by 64 (\(= 2^6\)). Quotient is the next digit.
- Divide remainder by 32 (\(= 2^5\)). Quotient is the next digit.
- Divide remainder by 16 (\(= 2^4\)). Quotient is the next digit.
- Divide remainder by 8 (\(= 2^3\)). Quotient is the next digit.
- Divide remainder by 4 (\(= 2^2\)). Quotient is the next digit.
- Divide remainder by 2 (\(= 2^1\)). Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...
2/64 is 0 rem 2. Next digit is 0: 10...
2/32 is 0 rem 2. Next digit is 0: 100...
2/16 is 0 rem 2. Next digit is 0: 1000...
2/8 is 0 rem 2. Next digit is 0: 10000...
2/4 is 0 remainder 2.
Decimal to Binary: Converting Between Bases

- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
  - Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
  - Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
  - The last remainder is the last digit.
- Example: what is 130 in binary notation?
  
  130/128 is 1 rem 2. First digit is 1: 1...
  
  2/64 is 0 rem 2. Next digit is 0: 10...
  
  2/32 is 0 rem 2. Next digit is 0: 100...
  
  2/16 is 0 rem 2. Next digit is 0: 1000...
  
  2/8 is 0 rem 2. Next digit is 0: 10000...
  
  2/4 is 0 remainder 2. Next digit is 0:
Decimal to Binary: Converting Between Bases

- From decimal to binary:
  - Divide by 128 (= $2^7$). Quotient is the first digit.
  - Divide remainder by 64 (= $2^6$). Quotient is the next digit.
  - Divide remainder by 32 (= $2^5$). Quotient is the next digit.
  - Divide remainder by 16 (= $2^4$). Quotient is the next digit.
  - Divide remainder by 8 (= $2^3$). Quotient is the next digit.
  - Divide remainder by 4 (= $2^2$). Quotient is the next digit.
  - Divide remainder by 2 (= $2^1$). Quotient is the next digit.
  - The last remainder is the last digit.
- Example: what is 130 in binary notation?
  
  130/128 is 1 rem 2. First digit is 1: 1...
  2/64 is 0 rem 2. Next digit is 0: 10...
  2/32 is 0 rem 2. Next digit is 0: 100...
  2/16 is 0 rem 2. Next digit is 0: 1000...
  2/8 is 0 rem 2. Next digit is 0: 10000...
  2/4 is 0 remainder 2. Next digit is 0: 100000...
Decimal to Binary: Converting Between Bases

From decimal to binary:
- Divide by 128 ($= 2^7$). Quotient is the first digit.
- Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?
  130/128 is 1 rem 2. First digit is 1: 1...
  2/64 is 0 rem 2. Next digit is 0: 10...
  2/32 is 0 rem 2. Next digit is 0: 100...
  2/16 is 0 rem 2. Next digit is 0: 1000...
  2/8 is 0 rem 2. Next digit is 0: 10000...
  2/4 is 0 remainder 2. Next digit is 0: 100000...
  2/2 is 1 rem 0.
From decimal to binary:

- Divide by 128 (\(= 2^7\)). Quotient is the first digit.
- Divide remainder by 64 (\(= 2^6\)). Quotient is the next digit.
- Divide remainder by 32 (\(= 2^5\)). Quotient is the next digit.
- Divide remainder by 16 (\(= 2^4\)). Quotient is the next digit.
- Divide remainder by 8 (\(= 2^3\)). Quotient is the next digit.
- Divide remainder by 4 (\(= 2^2\)). Quotient is the next digit.
- Divide remainder by 2 (\(= 2^1\)). Quotient is the next digit.
- The last remainder is the last digit.

Example: what is 130 in binary notation?

\[
\begin{align*}
130/128 & \text{ is 1 rem 2. First digit is 1: } 1\ldots \\
2/64 & \text{ is 0 rem 2. Next digit is 0: } 10\ldots \\
2/32 & \text{ is 0 rem 2. Next digit is 0: } 100\ldots \\
2/16 & \text{ is 0 rem 2. Next digit is 0: } 1000\ldots \\
2/8 & \text{ is 0 rem 2. Next digit is 0: } 10000\ldots \\
2/4 & \text{ is 0 remainder 2. Next digit is 0: } 100000\ldots \\
2/2 & \text{ is 1 rem 0. Next digit is 1: }
\end{align*}
\]
Decimal to Binary: Converting Between Bases

From decimal to binary:

- Divide by 128 ($= 2^7$). Quotient is the first digit.
- Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

$$\begin{align*}
130/128 & \text{ is 1 rem 2. First digit is 1: } 1… \\
2/64 & \text{ is 0 rem 2. Next digit is 0: } 10… \\
2/32 & \text{ is 0 rem 2. Next digit is 0: } 100… \\
2/16 & \text{ is 0 rem 2. Next digit is 0: } 1000… \\
2/8 & \text{ is 0 rem 2. Next digit is 0: } 10000… \\
2/4 & \text{ is 0 rem 2. Next digit is 0: } 100000… \\
2/2 & \text{ is 1 rem 0. Next digit is 1: } 1000001… \\
\end{align*}$$
From decimal to binary:

- Divide by 128 ($= 2^7$). Quotient is the first digit.
- Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- The last remainder is the last digit.

Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...
2/64 is 0 rem 2. Next digit is 0: 10...
2/32 is 0 rem 2. Next digit is 0: 100...
2/16 is 0 rem 2. Next digit is 0: 1000...
2/8 is 0 rem 2. Next digit is 0: 10000...
2/4 is 0 remainder 2. Next digit is 0: 100000...
2/2 is 1 rem 0. Next digit is 1: 1000001...
Adding the last remainder: 10000010
From decimal to binary:

- Divide by 128 ($= 2^7$). Quotient is the first digit.
- Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- The last remainder is the last digit.

**Example:** what is 130 in binary notation?

<table>
<thead>
<tr>
<th>Division</th>
<th>Quotient</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>130/128</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2/64</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2/32</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2/16</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2/8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2/4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2/2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Adding the last remainder:

10000010
Example: what is 99 in binary notation?

\[
\begin{align*}
\frac{99}{128} &= 0 \text{ rem } 99, \\
\frac{99}{64} &= 1 \text{ rem } 35, \\
\frac{35}{32} &= 1 \text{ rem } 3, \\
\frac{3}{16} &= 0 \text{ rem } 3, \\
\frac{3}{8} &= 0 \text{ rem } 3, \\
\frac{3}{4} &= 1 \text{ rem } 1,
\end{align*}
\]

Adding the last remainder:

\[01100011\]

Answer is 1100011.
Example: what is 99 in binary notation?

99/128 is 0 rem 99.
Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0:
Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...
99/64 is 1 rem 35.
Example: what is 99 in binary notation?

\[99/128 \text{ is } 0 \text{ rem } 99. \text{ First digit is } 0: \quad 0...\]

\[99/64 \text{ is } 1 \text{ rem } 35. \text{ Next digit is } 1: \]

Answer is 1100011.
Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...
99/64 is 1 rem 35. Next digit is 1: 01...
Decimal to Binary: Converting Between Bases

Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0:
99/64 is 1 rem 35. Next digit is 1:
35/32 is 1 rem 3.
Decimal to Binary: Converting Between Bases

Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...
99/64 is 1 rem 35. Next digit is 1: 01...
35/32 is 1 rem 3. Next digit is 1:
Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...
99/64 is 1 rem 35. Next digit is 1: 01...
35/32 is 1 rem 3. Next digit is 1: 011...
Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...
99/64 is 1 rem 35. Next digit is 1: 01...
35/32 is 1 rem 3. Next digit is 1: 011...
3/16 is 0 rem 3.
Decimal to Binary: Converting Between Bases

Example: what is 99 in binary notation?

\[
\begin{align*}
99/128 & \text{ is } 0 \text{ rem } 99. \text{ First digit is } 0: \quad 0... \\
99/64 & \text{ is } 1 \text{ rem } 35. \text{ Next digit is } 1: \quad 01... \\
35/32 & \text{ is } 1 \text{ rem } 3. \text{ Next digit is } 1: \quad 011... \\
3/16 & \text{ is } 0 \text{ rem } 3. \text{ Next digit is } 0:\n\end{align*}
\]
Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...
99/64 is 1 rem 35. Next digit is 1: 01...
35/32 is 1 rem 3. Next digit is 1: 011...
3/16 is 0 rem 3. Next digit is 0: 0110...
Example: what is 99 in binary notation?

\[
\begin{align*}
99/128 & \text{ is } 0 \text{ rem } 99. \text{ First digit is } 0: \quad 0 \ldots \\
99/64 & \text{ is } 1 \text{ rem } 35. \text{ Next digit is } 1: \quad 01 \ldots \\
35/32 & \text{ is } 1 \text{ rem } 3. \text{ Next digit is } 1: \quad 011 \ldots \\
3/16 & \text{ is } 0 \text{ rem } 3. \text{ Next digit is } 0: \quad 0110 \ldots \\
3/8 & \text{ is } 0 \text{ rem } 3.
\end{align*}
\]
Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0:   0...
99/64 is 1 rem 35. Next digit is 1:    01...
35/32 is 1 rem 3. Next digit is 1:     011...
3/16 is 0 rem 3. Next digit is 0:      0110...
3/8 is 0 rem 3. Next digit is 0:
Example: what is 99 in binary notation?

\[
\begin{align*}
99/128 &\text{ is 0 rem 99. First digit is 0: } \quad 0... \\
99/64 &\text{ is 1 rem 35. Next digit is 1: } \quad 01... \\
35/32 &\text{ is 1 rem 3. Next digit is 1: } \quad 011... \\
3/16 &\text{ is 0 rem 3. Next digit is 0: } \quad 0110... \\
3/8 &\text{ is 0 rem 3. Next digit is 0: } \quad 01100...
\end{align*}
\]
Example: what is 99 in binary notation?

\[
\begin{align*}
99/128 & \text{ is 0 rem 99. First digit is 0: } 0... \\
99/64 & \text{ is 1 rem 35. Next digit is 1: } 01... \\
35/32 & \text{ is 1 rem 3. Next digit is 1: } 011... \\
3/16 & \text{ is 0 rem 3. Next digit is 0: } 0110... \\
3/8 & \text{ is 0 rem 3. Next digit is 0: } 01100... \\
3/4 & \text{ is 0 remainder 3. }
\end{align*}
\]
Decimal to Binary: Converting Between Bases

Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...
99/64 is 1 rem 35. Next digit is 1: 01...
35/32 is 1 rem 3. Next digit is 1: 011...
3/16 is 0 rem 3. Next digit is 0: 0110...
3/8 is 0 rem 3. Next digit is 0: 01100...
3/4 is 0 remainder 3. Next digit is 0:
Decimal to Binary: Converting Between Bases

Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...
99/64 is 1 rem 35. Next digit is 1: 01...
35/32 is 1 rem 3. Next digit is 1: 011...
3/16 is 0 rem 3. Next digit is 0: 0110...
3/8 is 0 rem 3. Next digit is 0: 01100...
3/4 is 0 remainder 3. Next digit is 0: 011000...
Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...
99/64 is 1 rem 35. Next digit is 1: 01...
35/32 is 1 rem 3. Next digit is 1: 011...
3/16 is 0 rem 3. Next digit is 0: 0110...
3/8 is 0 rem 3. Next digit is 0: 01100...
3/4 is 0 remainder 3. Next digit is 0: 011000...
3/2 is 1 rem 1.
Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...
99/64 is 1 rem 35. Next digit is 1: 01...
35/32 is 1 rem 3. Next digit is 1: 011...
3/16 is 0 rem 3. Next digit is 0: 0110...
3/8 is 0 rem 3. Next digit is 0: 01100...
3/4 is 0 remainder 3. Next digit is 0: 011000...
3/2 is 1 rem 1. Next digit is 1:
Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...
99/64 is 1 rem 35. Next digit is 1: 01...
35/32 is 1 rem 3. Next digit is 1: 011...
3/16 is 0 rem 3. Next digit is 0: 0110...
3/8 is 0 rem 3. Next digit is 0: 01100...
3/4 is 0 remainder 3. Next digit is 0: 011000...
3/2 is 1 rem 1. Next digit is 1: 0110001...
Example: what is 99 in binary notation?

\[
\begin{align*}
99/128 & \text{ is 0 rem 99. First digit is 0: } 0... \\
99/64 & \text{ is 1 rem 35. Next digit is 1: } 01... \\
35/32 & \text{ is 1 rem 3. Next digit is 1: } 011... \\
3/16 & \text{ is 0 rem 3. Next digit is 0: } 0110... \\
3/8 & \text{ is 0 rem 3. Next digit is 0: } 01100... \\
3/4 & \text{ is 0 remainder 3. Next digit is 0: } 011000... \\
3/2 & \text{ is 1 rem 1. Next digit is 1: } 0110001... \\
\text{Adding the last remainder: } & 01100011
\end{align*}
\]
Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: \[ \ldots \]
99/64 is 1 rem 35. Next digit is 1: \[ 01 \ldots \]
35/32 is 1 rem 3. Next digit is 1: \[ 011 \ldots \]
3/16 is 0 rem 3. Next digit is 0: \[ 0110 \ldots \]
3/8 is 0 rem 3. Next digit is 0: \[ 01100 \ldots \]
3/4 is 0 remainder 3. Next digit is 0: \[ 011000 \ldots \]
3/2 is 1 rem 1. Next digit is 1: \[ 0110001 \ldots \]
Adding the last remainder: \[ 0110001 \]

Answer is 1100011.
From binary to decimal:
  ▶ Set sum = last digit.
Binary to Decimal: Converting Between Bases

- From binary to decimal:
  - Set sum = last digit.
  - Multiply next digit by \(2^n\). Add to sum.
  - Example: What is 111101 in decimal?
    - Sum starts with: 1
      - \(0 \times 2 = 0\). Add 0 to sum: 0
      - \(1 \times 4 = 4\). Add 4 to sum: 5
      - \(1 \times 8 = 8\). Add 8 to sum: 13
      - \(1 \times 16 = 16\). Add 16 to sum: 29
      - \(1 \times 32 = 32\). Add 32 to sum: 61
Binary to Decimal: Converting Between Bases

From binary to decimal:

- Set sum = last digit.
- Multiply next digit by $2 = 2^1$. Add to sum.
- Multiply next digit by $4 = 2^2$. Add to sum.

Example: What is $111101$ in decimal?

Sum starts with:
- $1 \times 2 = 0$. Add $0$ to sum:
- $1 \times 4 = 4$. Add $4$ to sum:
- $1 \times 8 = 8$. Add $8$ to sum:
- $1 \times 16 = 16$. Add $16$ to sum:
- $1 \times 32 = 32$. Add $32$ to sum:

Sum is the decimal number.
Binary to Decimal: Converting Between Bases

From binary to decimal:

- Set sum = last digit.
- Multiply next digit by $2^n$. Add to sum.
- Multiply next digit by $4 = 2^2$. Add to sum.
- Multiply next digit by $8 = 2^3$. Add to sum.

Example: What is 111101 in decimal?

Sum starts with:

1
0*2 = 0. Add 0 to sum:
1
1*4 = 4. Add 4 to sum:
5
1*8 = 8. Add 8 to sum:
13
1*16 = 16. Add 16 to sum:
29
1*32 = 32. Add 32 to sum:
61
Binary to Decimal: Converting Between Bases

- From binary to decimal:
  - Set sum = last digit.
  - Multiply next digit by $2 = 2^1$. Add to sum.
  - Multiply next digit by $4 = 2^2$. Add to sum.
  - Multiply next digit by $8 = 2^3$. Add to sum.
  - Multiply next digit by $16 = 2^4$. Add to sum.

Example: What is 111101 in decimal?

Sum starts with:
- $1 \times 2 = 0$. Add 0 to sum:
- $1 \times 4 = 4$. Add 4 to sum:
- $1 \times 8 = 8$. Add 8 to sum:
- $1 \times 16 = 16$. Add 16 to sum:
- $1 \times 32 = 32$. Add 32 to sum:

Sum is the decimal number:

61
Binary to Decimal: Converting Between Bases

From binary to decimal:

- Set sum = last digit.
- Multiply next digit by $2 = 2^1$. Add to sum.
- Multiply next digit by $4 = 2^2$. Add to sum.
- Multiply next digit by $8 = 2^3$. Add to sum.
- Multiply next digit by $16 = 2^4$. Add to sum.
- Multiply next digit by $32 = 2^5$. Add to sum.

Example: What is 111101 in decimal?

Sum starts with:

1
0*2 = 0. Add 0 to sum:
1
1*4 = 4. Add 4 to sum:
13
1*8 = 8. Add 8 to sum:
21
1*16 = 16. Add 16 to sum:
37
1*32 = 32. Add 32 to sum:
69

Sum is the decimal number.
From binary to decimal:

- Set sum = last digit.
- Multiply next digit by $2 = 2^1$. Add to sum.
- Multiply next digit by $4 = 2^2$. Add to sum.
- Multiply next digit by $8 = 2^3$. Add to sum.
- Multiply next digit by $16 = 2^4$. Add to sum.
- Multiply next digit by $32 = 2^5$. Add to sum.
- Multiply next digit by $64 = 2^6$. Add to sum.
From binary to decimal:

- Set sum = last digit.
- Multiply next digit by $2^1$. Add to sum.
- Multiply next digit by $2^2$. Add to sum.
- Multiply next digit by $2^3$. Add to sum.
- Multiply next digit by $2^4$. Add to sum.
- Multiply next digit by $2^5$. Add to sum.
- Multiply next digit by $2^6$. Add to sum.
- Multiply next digit by $2^7$. Add to sum.

Sum is the decimal number.

Example: What is 111101 in decimal?

Sum starts with:

- $1 \times 2^0 = 0$. Add 0 to sum:
- $1 \times 2^1 = 2$. Add 4 to sum:
- $1 \times 2^2 = 4$. Add 8 to sum:
- $1 \times 2^3 = 8$. Add 16 to sum:
- $1 \times 2^4 = 16$. Add 32 to sum:
- $1 \times 2^5 = 32$. Add 64 to sum:
- $1 \times 2^6 = 64$. Add 128 to sum:

Sum is 61.
Binary to Decimal: Converting Between Bases

From binary to decimal:

- Set sum = last digit.
- Multiply next digit by $2^1$. Add to sum.
- Multiply next digit by $2^2$. Add to sum.
- Multiply next digit by $2^3$. Add to sum.
- Multiply next digit by $2^4$. Add to sum.
- Multiply next digit by $2^5$. Add to sum.
- Multiply next digit by $2^6$. Add to sum.
- Multiply next digit by $2^7$. Add to sum.
- Sum is the decimal number.
From binary to decimal:

- Set sum = last digit.
- Multiply next digit by $2^1$. Add to sum.
- Multiply next digit by $2^2$. Add to sum.
- Multiply next digit by $2^3$. Add to sum.
- Multiply next digit by $2^4$. Add to sum.
- Multiply next digit by $2^5$. Add to sum.
- Multiply next digit by $2^6$. Add to sum.
- Multiply next digit by $2^7$. Add to sum.
- Sum is the decimal number.
- Example: What is 111101 in decimal?

Sum starts with:
Binary to Decimal: Converting Between Bases

From binary to decimal:

- Set sum = last digit.
- Multiply next digit by $2^1$. Add to sum.
- Multiply next digit by $2^2$. Add to sum.
- Multiply next digit by $2^3$. Add to sum.
- Multiply next digit by $2^4$. Add to sum.
- Multiply next digit by $2^5$. Add to sum.
- Multiply next digit by $2^6$. Add to sum.
- Multiply next digit by $2^7$. Add to sum.
- Sum is the decimal number.

Example: What is 111101 in decimal?

Sum starts with: 1
0*2 = 0. Add 0 to sum:
From binary to decimal:

- Set sum = last digit.
- Multiply next digit by $2 = 2^1$. Add to sum.
- Multiply next digit by $4 = 2^2$. Add to sum.
- Multiply next digit by $8 = 2^3$. Add to sum.
- Multiply next digit by $16 = 2^4$. Add to sum.
- Multiply next digit by $32 = 2^5$. Add to sum.
- Multiply next digit by $64 = 2^6$. Add to sum.
- Multiply next digit by $128 = 2^7$. Add to sum.
- Sum is the decimal number.

Example: What is 111101 in decimal?

Sum starts with: 1
0*2 = 0. Add 0 to sum: 1
Binary to Decimal: Converting Between Bases

From binary to decimal:

- Set sum = last digit.
- Multiply next digit by $2^1$. Add to sum.
- Multiply next digit by $2^2$. Add to sum.
- Multiply next digit by $2^3$. Add to sum.
- Multiply next digit by $2^4$. Add to sum.
- Multiply next digit by $2^5$. Add to sum.
- Multiply next digit by $2^6$. Add to sum.
- Multiply next digit by $2^7$. Add to sum.
- Sum is the decimal number.

Example: What is 111101 in decimal?

Sum starts with: 1
0*2 = 0. Add 0 to sum: 1
1*4 = 4. Add 4 to sum:
Binary to Decimal: Converting Between Bases

From binary to decimal:

- Set sum = last digit.
- Multiply next digit by $2^0$.
- Add to sum.
- Multiply next digit by $2^1$.
- Add to sum.
- Multiply next digit by $2^2$.
- Add to sum.
- Multiply next digit by $2^3$.
- Add to sum.
- Multiply next digit by $2^4$.
- Add to sum.
- Multiply next digit by $2^5$.
- Add to sum.
- Multiply next digit by $2^6$.
- Add to sum.
- Multiply next digit by $2^7$.
- Add to sum.

Example: What is 111101 in decimal?

<table>
<thead>
<tr>
<th>Sum starts with:</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0*2 = 0. Add 0 to sum:</td>
<td>1</td>
</tr>
<tr>
<td>1*4 = 4. Add 4 to sum:</td>
<td>5</td>
</tr>
</tbody>
</table>
From binary to decimal:

- Set sum = last digit.
- Multiply next digit by $2 = 2^1$. Add to sum.
- Multiply next digit by $4 = 2^2$. Add to sum.
- Multiply next digit by $8 = 2^3$. Add to sum.
- Multiply next digit by $16 = 2^4$. Add to sum.
- Multiply next digit by $32 = 2^5$. Add to sum.
- Multiply next digit by $64 = 2^6$. Add to sum.
- Multiply next digit by $128 = 2^7$. Add to sum.
- Sum is the decimal number.
- Example: What is 111101 in decimal?

  Sum starts with: 1
  0*2 = 0. Add 0 to sum: 1
  1*4 = 4. Add 4 to sum: 5
  1*8 = 8. Add 8 to sum:
From binary to decimal:

- Set sum = last digit.
- Multiply next digit by $2 = 2^1$. Add to sum.
- Multiply next digit by $4 = 2^2$. Add to sum.
- Multiply next digit by $8 = 2^3$. Add to sum.
- Multiply next digit by $16 = 2^4$. Add to sum.
- Multiply next digit by $32 = 2^5$. Add to sum.
- Multiply next digit by $64 = 2^6$. Add to sum.
- Multiply next digit by $128 = 2^7$. Add to sum.
- Sum is the decimal number.

**Example:** What is $111101$ in decimal?

<table>
<thead>
<tr>
<th>Sum starts with:</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0 \times 2$ = 0</td>
<td>Add 0 to sum:</td>
</tr>
<tr>
<td>$1 \times 4$ = 4</td>
<td>Add 4 to sum:</td>
</tr>
<tr>
<td>$1 \times 8$ = 8</td>
<td>Add 8 to sum:</td>
</tr>
</tbody>
</table>
From binary to decimal:

- Set sum = last digit.
- Multiply next digit by $2 = 2^1$. Add to sum.
- Multiply next digit by $4 = 2^2$. Add to sum.
- Multiply next digit by $8 = 2^3$. Add to sum.
- Multiply next digit by $16 = 2^4$. Add to sum.
- Multiply next digit by $32 = 2^5$. Add to sum.
- Multiply next digit by $64 = 2^6$. Add to sum.
- Multiply next digit by $128 = 2^7$. Add to sum.
- Sum is the decimal number.

Example: What is 111101 in decimal?

Sum starts with: 1
0*2 = 0. Add 0 to sum: 1
1*4 = 4. Add 4 to sum: 5
1*8 = 8. Add 8 to sum: 13
1*16 = 16. Add 16 to sum: 29
Binary to Decimal: Converting Between Bases

From binary to decimal:

- Set sum = last digit.
- Multiply next digit by 2 = $2^1$. Add to sum.
- Multiply next digit by 4 = $2^2$. Add to sum.
- Multiply next digit by 8 = $2^3$. Add to sum.
- Multiply next digit by 16 = $2^4$. Add to sum.
- Multiply next digit by 32 = $2^5$. Add to sum.
- Multiply next digit by 64 = $2^6$. Add to sum.
- Multiply next digit by 128 = $2^7$. Add to sum.
- Sum is the decimal number.

Example: What is 111101 in decimal?

Sum starts with: 1
0*2 = 0. Add 0 to sum: 1
1*4 = 4. Add 4 to sum: 5
1*8 = 8. Add 8 to sum: 13
1*16 = 16. Add 16 to sum: 29
Binary to Decimal: Converting Between Bases

From binary to decimal:

- Set sum = last digit.
- Multiply next digit by $2 = 2^1$. Add to sum.
- Multiply next digit by $4 = 2^2$. Add to sum.
- Multiply next digit by $8 = 2^3$. Add to sum.
- Multiply next digit by $16 = 2^4$. Add to sum.
- Multiply next digit by $32 = 2^5$. Add to sum.
- Multiply next digit by $64 = 2^6$. Add to sum.
- Multiply next digit by $128 = 2^7$. Add to sum.
- Sum is the decimal number.

Example: What is 111101 in decimal?

Sum starts with: 1
0*2 = 0. Add 0 to sum: 1
1*4 = 4. Add 4 to sum: 5
1*8 = 8. Add 8 to sum: 13
1*16 = 16. Add 16 to sum: 29
1*32 = 32. Add 32 to sum:
Binary to Decimal: Converting Between Bases

From binary to decimal:

- Set sum = last digit.
- Multiply next digit by $2 = 2^1$. Add to sum.
- Multiply next digit by $4 = 2^2$. Add to sum.
- Multiply next digit by $8 = 2^3$. Add to sum.
- Multiply next digit by $16 = 2^4$. Add to sum.
- Multiply next digit by $32 = 2^5$. Add to sum.
- Multiply next digit by $64 = 2^6$. Add to sum.
- Multiply next digit by $128 = 2^7$. Add to sum.
- Sum is the decimal number.
- Example: What is 111101 in decimal?

Sum starts with: 1
- 0*2 = 0. Add 0 to sum: 1
- 1*4 = 4. Add 4 to sum: 5
- 1*8 = 8. Add 8 to sum: 13
- 1*16 = 16. Add 16 to sum: 29
- 1*32 = 32. Add 32 to sum: 61
From binary to decimal:

- Set sum = last digit.
- Multiply next digit by $2 = 2^1$. Add to sum.
- Multiply next digit by $4 = 2^2$. Add to sum.
- Multiply next digit by $8 = 2^3$. Add to sum.
- Multiply next digit by $16 = 2^4$. Add to sum.
- Multiply next digit by $32 = 2^5$. Add to sum.
- Multiply next digit by $64 = 2^6$. Add to sum.
- Multiply next digit by $128 = 2^7$. Add to sum.
- Sum is the decimal number.
- Example: What is 111101 in decimal?

Sum starts with: 1
0*2 = 0. Add 0 to sum: 1
1*4 = 4. Add 4 to sum: 5
1*8 = 8. Add 8 to sum: 13
1*16 = 16. Add 16 to sum: 29
1*32 = 32. Add 32 to sum: 61
Example: What is 10100100 in decimal?

Sum starts with:
Example: What is 10100100 in decimal?

Sum starts with: 0
0*2 = 0. Add 0 to sum:

The answer is 164.
Example: What is 10100100 in decimal?

Sum starts with: 0
0*2 = 0. Add 0 to sum: 0
0*4 = 0. Add 0 to sum: 0
0*8 = 0. Add 0 to sum: 0
1*16 = 16. Add 16 to sum: 16
0*32 = 0. Add 0 to sum: 16
1*64 = 64. Add 64 to sum: 80
0*128 = 0. Add 0 to sum: 80
The answer is 80.
Example: What is 10100100 in decimal?

Sum starts with: 0

0*2 = 0. Add 0 to sum: 0

1*4 = 4. Add 4 to sum:
Example: What is 10100100 in decimal?

- Sum starts with: 0
- 0*2 = 0. Add 0 to sum: 0
- 1*4 = 4. Add 4 to sum: 4

The answer is 164.
Binary to Decimal: Converting Between Bases

Example: What is 10100100 in decimal?

Sum starts with: 0
0*2 = 0. Add 0 to sum: 0
1*4 = 4. Add 4 to sum: 4
0*8 = 0. Add 0 to sum: 4

The answer is 164.
Example: What is 10100100 in decimal?

Sum starts with: 0
0*2 = 0. Add 0 to sum: 0
1*4 = 4. Add 4 to sum: 4
0*8 = 0. Add 0 to sum: 4

The answer is 164.
Example: What is 10100100 in decimal?

Sum starts with: 0
0*2 = 0. Add 0 to sum: 0
1*4 = 4. Add 4 to sum: 4
0*8 = 0. Add 0 to sum: 4
0*16 = 0. Add 0 to sum:

The answer is 164.
Example: What is 10100100 in decimal?

Sum starts with: 0
0*2 = 0. Add 0 to sum: 0
1*4 = 4. Add 4 to sum: 4
0*8 = 0. Add 0 to sum: 4
0*16 = 0. Add 0 to sum: 4

The answer is 164.
Example: What is 10100100 in decimal?

Sum starts with: 0
0*2 = 0. Add 0 to sum: 0
1*4 = 4. Add 4 to sum: 4
0*8 = 0. Add 0 to sum: 4
0*16 = 0. Add 0 to sum: 4
1*32 = 32. Add 32 to sum: 36
Example: What is 10100100 in decimal?

Sum starts with: 0
0*2 = 0. Add 0 to sum: 0
1*4 = 4. Add 4 to sum: 4
0*8 = 0. Add 0 to sum: 4
0*16 = 0. Add 0 to sum: 4
1*32 = 32. Add 32 to sum: 36

The answer is 164.
Example: What is 10100100 in decimal?

Sum starts with: 0
0*2 = 0. Add 0 to sum: 0
1*4 = 4. Add 4 to sum: 4
0*8 = 0. Add 0 to sum: 4
0*16 = 0. Add 0 to sum: 4
1*32 = 32. Add 32 to sum: 36
0*64 = 0. Add 0 to sum: 36

The answer is 164.
Example: What is 10100100 in decimal?

Sum starts with: 0
0*2 = 0. Add 0 to sum: 0
1*4 = 4. Add 4 to sum: 4
0*8 = 0. Add 0 to sum: 4
0*16 = 0. Add 0 to sum: 4
1*32 = 32. Add 32 to sum: 36
0*64 = 0. Add 0 to sum: 36

The answer is 164.
Example: What is 10100100 in decimal?

Sum starts with: 0
0*2 = 0. Add 0 to sum: 0
1*4 = 4. Add 4 to sum: 4
0*8 = 0. Add 0 to sum: 4
0*16 = 0. Add 0 to sum: 4
1*32 = 32. Add 32 to sum: 36
0*64 = 0. Add 0 to sum: 36
1*128 = 0. Add 128 to sum: 164.

The answer is 164.
Example: What is 10100100 in decimal?

Sum starts with: 0
0*2 = 0. Add 0 to sum: 0
1*4 = 4. Add 4 to sum: 4
0*8 = 0. Add 0 to sum: 4
0*16 = 0. Add 0 to sum: 4
1*32 = 32. Add 32 to sum: 36
0*64 = 0. Add 0 to sum: 36
1*128 = 0. Add 128 to sum: 164
Example: What is 10100100 in decimal?

Sum starts with: 0
0*2 = 0. Add 0 to sum: 0
1*4 = 4. Add 4 to sum: 4
0*8 = 0. Add 0 to sum: 4
0*16 = 0. Add 0 to sum: 4
1*32 = 32. Add 32 to sum: 36
0*64 = 0. Add 0 to sum: 36
1*128 = 0. Add 128 to sum: 164

The answer is 164.
Design Challenge: Incrementers

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

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

```python
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"
Design Challenge: Incrementers

- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:
  
  ```python
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"
Design Challenge: Incrementers

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

Example: Increment a decimal number:

```python
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"
```
Design Challenge: Incrementers

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

Example: Increment a decimal number:

```python
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.
Design Challenge: Incrementers

- Simplest arithmetic: add one (“increment”) a variable.
- Example: Increment a decimal number:
  ```python
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.*
Design Challenge: Incrementers

- Simplest arithmetic: add one (“increment”) a variable.
- Example: Increment a decimal number:
  ```python
  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.
Design Challenge: Incrementers

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

- Searching through data is a common task—built-in functions and standard design patterns for this.
Recap

- Searching through data is a common task—built-in functions and standard design patterns for this.
- Programming languages can be classified by the level of abstraction and direct access to data.
Recap

- Searching through data is a common task—built-in functions and standard design patterns for this.
- Programming languages can be classified by the level of abstraction and direct access to data.
Today’s Topics

- Design Patterns: Searching
- Python Recap
- Machine Language
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- Final Exam: Format
Final Overview: Administration

- The exam will be administered through Gradescope.
Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on **May 18, 9am-10:30am**
Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on **May 18, 9am-10:30am**
- There will be a different Gradescope Course called **CSci 127 Final Exam**
Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on **May 18, 9am-10:30am**
- There will be a different Gradescope Course called **CSci 127 Final Exam**
- Prior to the exam you will be added to the final exam course for your exam version.
Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on May 18, 9am-10:30am
- There will be a different Gradescope Course called CSci 127 Final Exam
- Prior to the exam you will be added to the final exam course for your exam version.
- The only assignment in that course will be your final exam.
Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on **May 18, 9am-10:30am**
- There will be a different Gradescope Course called **CSci 127 Final Exam**
- Prior to the exam you will be added to the final exam course for your exam version.
- The only assignment in that course will be your final exam.
- The morning of the exam: log into Gradescope, find the **CSci 127 Final Exam** course and open the assignment.
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).
Exam Options

Exam Times:

Final Exam, Version 3
CSci 127: Introduction to Computer Science
Hunter College, City University of New York

Exam Rules:

• Show all your work. Your grade will be based on the work shown.
• The exam is closed book and closed notes, with the exception of a 8 1/2" x 11" piece of paper filled with notes, programs, etc.
• When taking the exam, you must have with you pens and pencils, and your note sheet.
• You may not use a computer, calculator, tablet, smart watch, or other electronic device.
• Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating on examinations, obtaining unfair advantage, and falsification of records and official documents) as serious offenses against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

I understand that all cases of academic dishonesty will be reported to the Dean of Students and will result in sanctions.

Name:
EmpID:
Email:
Signature:
Exam Options

Exam Times:

- Default: Regular Time: Monday, 18 May, 9-10:30am.
- Alternate Time: Reading Day, Friday, 15 May, 8:00am-9:30am.
- Accessibility Testing: For double time must contact Prof. Ligorio by 15 May.
Exam Rules:

- Show all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of an 8.5" x 11" piece of paper filled with notes, programs, etc.
- You may not use a computer, calculator, tablet, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating on examinations, obtaining unfair advantage, and falsification of records and other official documents) as serious offenses against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

I understand that all cases of academic dishonesty will be reported to the Dean of Students and will result in sanctions.

Name:
EmpID:
Email:
Signature:
Exam Options

Exam Times:

- **Default**: Regular Time: Monday, 18 May, 9-10:30am.
- **Alternate Time**: Reading Day, Friday, 15 May, 8:00am-9:30am.
- **Accessibility Testing**: For double time must contact Prof. Ligorio by 15 May.

Grading Options:

- **Default**: Letter Grade.
- **Credit/NoCredit grade**— availability depends on major and academic standing.
Exam Options

Exam Times:

- **Default:** Regular Time: Monday, 18 May, 9-10:30am.
- **Alternate Time:** Reading Day, Friday, 15 May, 8:00am-9:30am.
- **Accessibility Testing:** For double time must contact Prof. Ligorio by 15 May.

Grading Options:

- **Default:** Letter Grade.
- **Credit/NoCredit grade**— availability depends on major and academic standing.

Survey for your choices will be available next lecture.
Exam Options

Exam Times:

- **Default:** Regular Time: Monday, 18 May, 9-10:30am.
- **Alternate Time:** Reading Day, Friday, 15 May, 8:00am-9:30am.
- **Accessibility Testing:** For double time must contact Prof. Ligorio by 15 May.

Grading Options:

- **Default:** Letter Grade.
- **Credit/NoCredit grade**— availability depends on major and academic standing.

Survey for your choices will be available next lecture.

No survey answer implies you will take the exam on 18 May, regular time and letter grade.