

CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

- This lecture will be recorded

Announcements

- Thanksgiving Break starts in 9 days.



Announcements

- Thanksgiving Break starts in 9 days.
- No CUNY classes:
Thursday-Saturday, 26-29 November.



Announcements



- Thanksgiving Break starts in 9 days.
- No CUNY classes:
Thursday-Saturday, 26-29 November.
- Add my email to your contacts and check your spam folders. I reply within 24/48 hours at most (not on weekends).

Announcements



- Thanksgiving Break starts in 9 days.
- No CUNY classes:
Thursday-Saturday, 26-29 November.
- Add my email to your contacts and check your spam folders. I reply within 24/48 hours at most (not on weekends).
- In response to wrap-up requests, additional challenges today with while loops and binary & hexadecimal numbers.

Frequently Asked Questions

From email and tutoring.

- **Help! Binary & hexadecimal numbers make no sense!**

Frequently Asked Questions

From email and tutoring.

- Help! Binary & hexadecimal numbers make no sense!
No worries— we'll start off with those in today's lecture.

Frequently Asked Questions

From email and tutoring.

- **Help! Binary & hexadecimal numbers make no sense!**
No worries— we'll start off with those in today's lecture.
- **When is the final? Is there a review sheet?**

Frequently Asked Questions

From email and tutoring.

- **Help! Binary & hexadecimal numbers make no sense!**
No worries— we'll start off with those in today's lecture.
- **When is the final? Is there a review sheet?**
The official final is Monday, 14 December, 9-11am.

Frequently Asked Questions

From email and tutoring.

- **Help! Binary & hexadecimal numbers make no sense!**

No worries— we'll start off with those in today's lecture.

- **When is the final? Is there a review sheet?**

The official final is Monday, 14 December, 9-11am.

The early final exam (alternative date) is on Friday, 11 December (Reading Day), 8-10am.

Frequently Asked Questions

From email and tutoring.

- **Help! Binary & hexadecimal numbers make no sense!**

No worries— we'll start off with those in today's lecture.

- **When is the final? Is there a review sheet?**

The official final is Monday, 14 December, 9-11am.

The early final exam (alternative date) is on Friday, 11 December (Reading Day), 8-10am.

Instead of a review sheet, we have:

Frequently Asked Questions

From email and tutoring.

- **Help! Binary & hexadecimal numbers make no sense!**

No worries— we'll start off with those in today's lecture.

- **When is the final? Is there a review sheet?**

The official final is Monday, 14 December, 9-11am.

The early final exam (alternative date) is on Friday, 11 December (Reading Day), 8-10am.

Instead of a review sheet, we have:

- ▶ *All previous final exams (and answer keys) on the website.*

Frequently Asked Questions

From email and tutoring.

- **Help! Binary & hexadecimal numbers make no sense!**

No worries— we'll start off with those in today's lecture.

- **When is the final? Is there a review sheet?**

The official final is Monday, 14 December, 9-11am.

The early final exam (alternative date) is on Friday, 11 December (Reading Day), 8-10am.

Instead of a review sheet, we have:

- ▶ *All previous final exams (and answer keys) on the website.*
- ▶ *UTAs in drop-in tutoring happy to review concepts and old exam questions.*

Frequently Asked Questions

From email and tutoring.

- **Help! Binary & hexadecimal numbers make no sense!**

No worries— we'll start off with those in today's lecture.

- **When is the final? Is there a review sheet?**

The official final is Monday, 14 December, 9-11am.

The early final exam (alternative date) is on Friday, 11 December (Reading Day), 8-10am.

Instead of a review sheet, we have:

- ▶ *All previous final exams (and answer keys) on the website.*
- ▶ *UTAs in drop-in tutoring happy to review concepts and old exam questions.*
- ▶ *There will be opportunity for some practice and to ask review questions during our last meeting on 8 December.*

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:

```
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.')|
```

Python Tutor

```
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.')
```

(Demo with pythonTutor)

Design Pattern: Linear Search

```
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.')
```

- Example of **linear search**.

Design Pattern: Linear Search

```
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.')
```

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

Design Pattern: Linear Search

```
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.')
```

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

Design Pattern: Linear Search

```
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.')
```

- 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
```

← These lines are comments

```
#Date: September 1, 2017
```

← (for us, not computer to read)

```
#This program prints: Hello, World!
```

← (this one also)

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

← Prints the string "Hello, World!" to the screen

Week 1: print(), loops, comments, & turtles

- Introduced comments & print():

```
#Name: Thomas Hunter
```

← These lines are comments

```
#Date: September 1, 2017
```

← (for us, not computer to read)

```
#This program prints: Hello, World!
```

← (this one also)

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

← Prints the string "Hello, World!" to the screen

- As well as definite loops & the turtle package:

The screenshot shows a code editor interface with a toolbar at the top. The file tab shows 'main.py'. The code area contains the following Python script:

```
1 #A program that demonstrates turtles stamping
2
3 import turtle
4
5 taylor = turtle.Turtle()
6 taylor.color("purple")
7 taylor.shape("turtle")
8
9 for i in range(6):
10     taylor.forward(100)
11     taylor.stamp()
12     taylor.left(60)
```

To the right of the code editor is a results panel titled 'Result' which displays a purple hexagon drawn by the turtle. Each vertex of the hexagon has a small purple star-like stamp.

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

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

- 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

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

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

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.

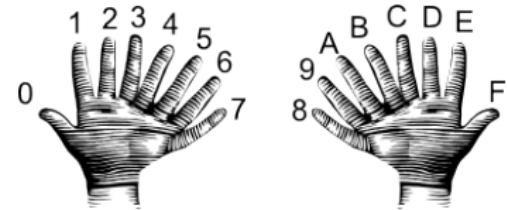
Week 2: variables, data types, more on loops & range()

- 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.
- More on loops & ranges:

```
1 #Predict what will be printed:  
2  
3 for num in [2,4,6,8,10]:  
4     print(num)  
5  
6 sum = 0  
7 for x in range(0,12,2):  
8     print(x)  
9     sum = sum + x  
10  
11 print(sum)  
12  
13 for c in "ABCD":  
14     print(c)
```

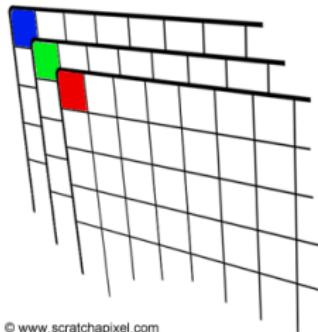
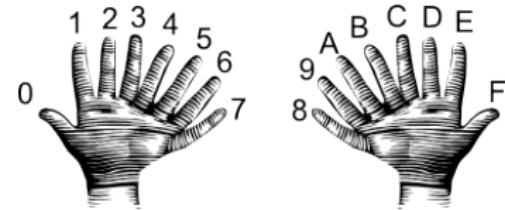
Week 3: colors, hex, slices, numpy & images

Color Name	HEX	Color
Black	#000000	
Navy	#000080	
DarkBlue	#00008B	
MediumBlue	#0000CD	
Blue	#0000FF	



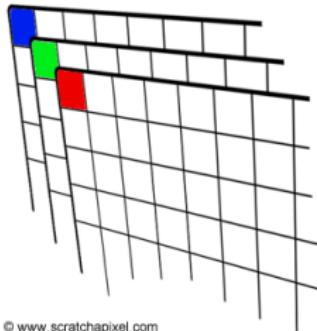
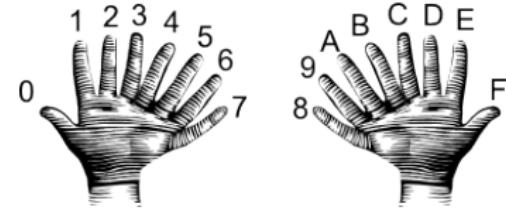
Week 3: colors, hex, slices, numpy & images

Color Name	HEX	Color
Black	#000000	
Navy	#000080	
DarkBlue	#00008B	
MediumBlue	#0000CD	
Blue	#0000FF	



Week 3: colors, hex, slices, numpy & images

Color Name	HEX	Color
Black	#000000	
Navy	#000080	
DarkBlue	#00008B	
MediumBlue	#0000CD	
Blue	#0000FF	



```
>>> 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]])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

Week 4: design problem (cropping images) & decisions



Week 4: design problem (cropping images) & decisions



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

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.

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

```
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

```
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.")

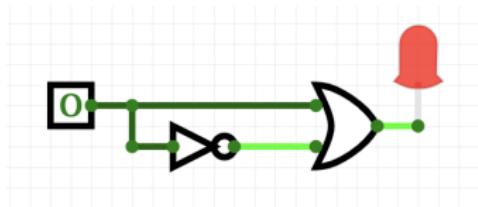
visibility = 0.2
winds = 40
conditions = "blowing snow"
if (winds > 35) and (visibility < 0.25) and \
    (conditions == "blowing snow" or conditions == "heavy snow"):
    print("Blizzard!")
```

Week 5: logical operators, truth tables & logical circuits

```
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!")
```

in1	and	in2	returns:
False	and	False	False
False	and	True	False
True	and	False	False
True	and	True	True



Week 6: structured data, pandas, & more design

```
Source: https://en.wikipedia.org/wiki/Demographics_of_New_York_City.....  
All population figures are consistent with present-day boundaries.....  
First census after the consolidation of the five boroughs.....  
.....  
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total  
1690,1,037,2037,727,788,2037  
1771,21843,36232,,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,75955  
1810,67534,5854,6442,1755,4543,75934  
1820,123704,11187,8246,2792,6135,152056  
1830,202589,20535,9049,3023,7082,242278  
1840,312110,11013,14035,5346,10965,391114  
1850,35344,12800,18951,5346,10965,391115  
1860,813469,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59945,5653,51980,33029,1911801  
1890,1367711,66411,6348,51980,33029,1911804  
1900,1850593,116582,152999,200567,67621,2437202  
1910,2233142,1634351,264041,430980,8569,4766803  
1920,2211103,2018354,446031,446031,73201,11651,59148  
1930,1867128,1867128,1867128,1867128,1867128,4930446  
1940,1889924,2469285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550949,1451277,191555,7891957  
1960,1690311,2039239,1899341,1471701,2039239,7881984  
1970,1539231,2460705,1471701,1471701,135443,7881984  
1980,1426285,2230936,1891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1203789,378977,7322564  
2000,1537195,2485326,2229379,1332450,419782,8080879  
2010,1583873,2504705,2272722,1385108,447512,8175133  
2015,1444518,2636733,2339150,1459446,474558,8059405
```

nycHistPop.csv

In Lab 6

Week 6: structured data, pandas, & more design

```
import matplotlib.pyplot as plt  
import pandas as pd
```

Source: https://en.wikipedia.org/wiki/Demographics_of_New_York_City,
All population figures are consistent with present-day boundaries.....
Five census after the consolidation of the five boroughs.....
.....
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total
1890,4937,2037,,727,7881,28423
1870,33131,4549,6159,1781,3827,49447
1860,60515,5740,6442,1755,4543,75955
1850,55545,5254,5851,1541,3973,74934
1820,123704,11187,8246,2792,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312110,18013,14031,5348,10965,391114
1850,35545,21891,18591,5348,10965,391114
1860,813469,279122,32903,23593,25492,174777
1870,942292,419921,45468,37393,33029,1479103
1880,1164473,59945,5653,51980,33029,1911803
1890,1367711,70001,6534,5816,33029,1911803
1900,185093,116582,152999,200567,67921,2437202
1910,2233142,1634351,284041,430980,8569,4766803
1920,2210103,2018354,44601,44601,73201,11651,500488
1930,1667137,1796128,1796128,1796128,1796128,4930446
1940,1889924,2469285,1297634,1394711,174441,7454995
1950,1960101,2738175,1550949,1451277,191555,7991957
1960,1690101,2319175,1809049,1451277,191555,7981984
1970,1539231,2460701,1471701,1471701,135443,7981984
1980,1426285,2230936,1891325,1168972,352121,7071639
1990,1487536,2300664,1951598,1203789,379977,7322564
2000,1537195,2485326,2223379,1332450,419728,8080879
2010,1583873,2504705,2217722,1385108,4175133,8175133
2015,1444018,2646733,2339150,1459444,474558,8059405

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)
```

```
Source: https://en.wikipedia.org/wiki/Demographics_of_New_York_City.....  
All population figures are consistent with present-day boundaries.....  
First census after the consolidation of the five boroughs.....  
.....  
Year,Borough,Brooklyn,Queens,Bronx,Staten Island,Total  
1690,4937,2017,...,727,7181  
1771,21843,36232,...,2847,28423  
1790,33131,4549,...,6159,1781,3827,49447  
1800,60515,5740,...,6442,1755,4543,75955  
1810,69111,6210,...,6442,1755,4543,75934  
1820,123704,11187,...,8246,2792,6135,152056  
1830,20589,20535,...,9049,3023,7082,242278  
1840,31150,10113,...,14075,5346,10965,391114  
1850,35549,...,12895,...,14851,5346,10965,391115  
1860,513469,...,279122,...,23993,...,25492,174777  
1870,942292,...,419921,...,45468,...,37393,...,33029,...,1479103  
1880,1164473,...,59943,...,5653,...,51980,...,39301,...,1911801  
1890,1360511,...,79111,...,65346,...,51980,...,39301,...,1911804  
1900,185093,...,116582,...,152999,...,200567,...,67921,...,2437202  
1910,233142,...,1634351,...,2841,...,430980,...,8569,...,476683  
1920,2241103,...,2018354,...,44601,...,72021,...,11651,...,593083  
1930,2667111,...,2079128,...,44601,...,72021,...,11651,...,5930446  
1940,...,1889924,...,2690285,...,1297634,...,1394711,...,174441,...,7454995  
1950,...,1960101,...,2738175,...,1550949,...,1451277,...,191555,...,7991957  
1960,...,1660101,...,2738175,...,1550949,...,1451277,...,191555,...,7991957  
1970,...,1659331,...,2465701,...,1472701,...,135443,...,7071646  
1980,...,1426285,...,2230936,...,1891325,...,1168972,...,352121,...,7071639  
1990,...,1487536,...,2300664,...,1951598,...,1302789,...,379977,...,7322564  
2000,...,1537195,...,2485326,...,2229379,...,1332650,...,419728,...,8080879  
2010,...,1583873,...,2504705,...,2272722,...,1385108,...,474558,...,8175133  
2015,...,1444918,...,2546733,...,2339150,...,1459446,...,474558,...,8159405
```

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)
```

```
Source: https://en.wikipedia.org/wiki/Demographics_of_New_York_City.....  
All population figures are consistent with present-day boundaries.....  
First census after the consolidation of the five boroughs.....  
.....  
Year,Population  
1690,203,2037,...,727,7181  
1771,21843,36231,...,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,75955  
1810,72000,6350,7000,1800,5000,89734  
1820,123704,11187,8246,2792,6135,152056  
1830,20589,20535,9049,3023,7082,242278  
1840,31510,19113,14000,5348,10965,391114  
1850,35549,21890,18800,5800,11000,45115  
1860,813469,279122,23903,23933,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59943,5653,51980,33091,1911801  
1890,1385000,720000,68000,51800,35000,210000  
1900,1850093,116582,152999,200567,67621,2437202  
1910,2233142,1634351,2841,430980,8569,476683  
1920,22161103,2018354,44600,720201,11650,50000  
1930,2667112,2079128,1079128,1079128,1079128,4930446  
1940,1889924,2690285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550949,1451277,191555,7991957  
1960,1690000,2319319,1890000,1460000,191555,7981984  
1970,1539231,2460701,204731,1472701,135443,7984640  
1980,1426285,2230936,1891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1203789,378977,7322564  
2000,1537195,2485326,2223379,1332450,419782,8080879  
2010,1583873,2504705,2217722,1385108,419782,8175133  
2015,1444018,2640733,2339150,1459446,474558,8059405
```

nycHistPop.csv

In Lab 6

Week 6: structured data, pandas, & more design

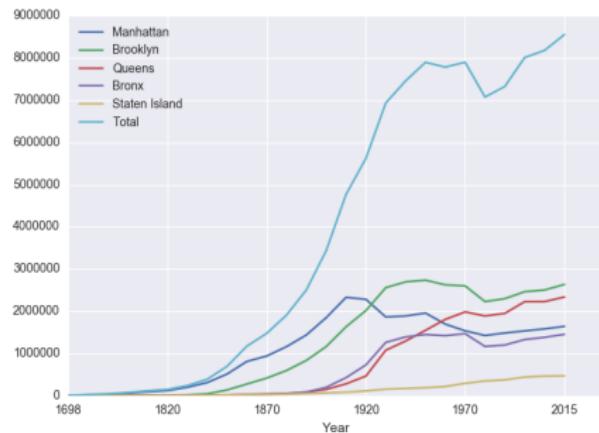
```
import matplotlib.pyplot as plt  
import pandas as pd
```

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

```
Source: https://en.wikipedia.org/wiki/Demographics_of_New_York_City.....  
All population figures are consistent with present-day boundaries.....  
First census after the consolidation of the five boroughs.....  
.....  
Year,Borough,Population  
1699,Manhattan, Brooklyn, Queens, Bronx, Staten Island, Total  
1771,21843,36231,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,75955  
1810,70531,6354,7041,1803,4937,93734  
1820,123704,11187,8246,2792,6135,152056  
1830,202589,20535,9049,3023,7082,242278  
1840,312110,18013,14041,5348,10965,391114  
1850,355441,21800,18500,5850,13000,45115  
1860,813469,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59943,5653,51980,33051,1911801  
1890,1384473,72000,6000,58000,35000,2151514  
1900,1850093,116582,152999,200507,67921,2437202  
1910,233142,1634351,2841,430980,8569,476683  
1920,2210103,2018354,44607,73201,11651,50048  
1930,2667103,2485254,579128,125245,15837,630446  
1940,1889924,2690285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550849,1451277,191555,7091957  
1960,1696010,2738175,1550849,1451277,191555,7091957  
1970,1639231,2465701,1472701,1247101,135443,708460  
1980,1426285,2230936,11891325,1168972,352121,7071639  
1990,1497536,2300664,1951598,1320789,378977,7322564  
2000,1537195,2485326,2229379,1332450,413728,8080879  
2010,1583873,2504705,2216722,1385108,413728,8175133  
2015,1444518,2636733,2339150,1459446,474558,8059405
```

nycHistPop.csv

In Lab 6



Week 7: functions

- 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()
```

Week 7: functions

```
#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.

Week 7: functions

```
#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.
- The opening function is often called `main()`

Week 7: functions

```
#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.
- The opening function is often called `main()`
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:

Week 7: functions

```
#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.
- 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")`

Week 7: functions

```
#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.
- 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,

Week 7: functions

```
#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.
- 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**.

```
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)
```

Week 8: function parameters, github

```
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).

Week 8: function parameters, github

```
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

```
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**

Week 8: function parameters, github

```
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

```
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)
                                Actual Parameters

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 9: top-down design, folium, loops, and random()



```
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()

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

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

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

```
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()

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

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

- 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.

```
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()

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

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

```
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.

Week 10: more on loops, max design pattern, random()

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

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

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

Week 10: more on loops, max design pattern, random()

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

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

```
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.
- To use, must include:
`import random`.
- The max design pattern provides a template for finding maximum value from a list.

Python & Circuits Review: 10 Weeks in 10 Minutes



- Input/Output (I/O): `input()` and `print()`; pandas for CSV files
- Types:
 - ▶ Primitive: `int`, `float`, `bool`, `string`;
 - ▶ Container: lists (but not dictionaries/hashes 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`

Lecture Quiz

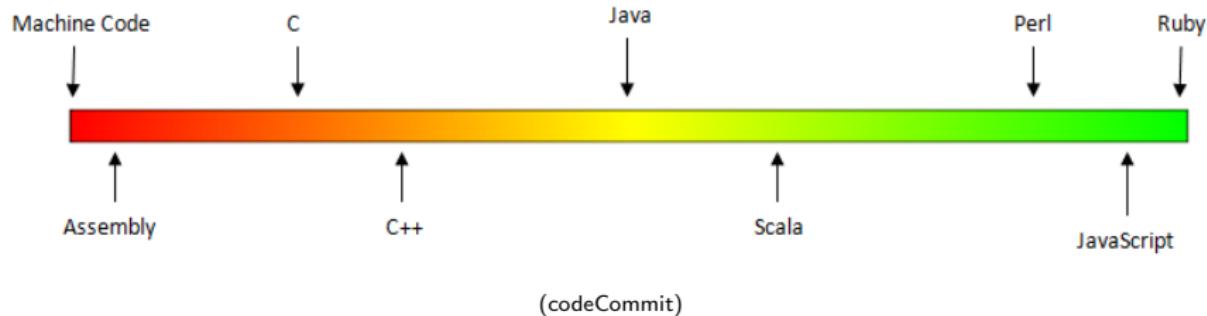
- Log-in to Gradescope
- Find LECTURE 11 Quiz
- Take the quiz
- **You have 3 minutes**

Today's Topics



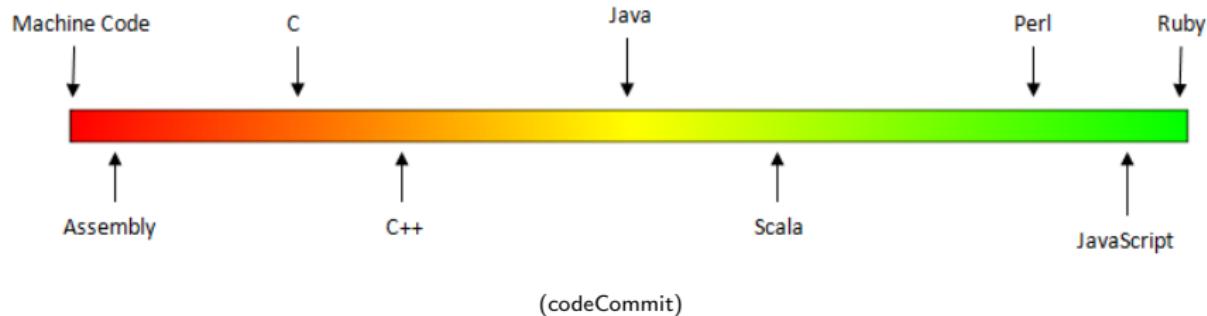
- Design Patterns: Searching
- Python Recap
- **Machine Language**
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- Final Exam: Format

Low-Level vs. High-Level Languages



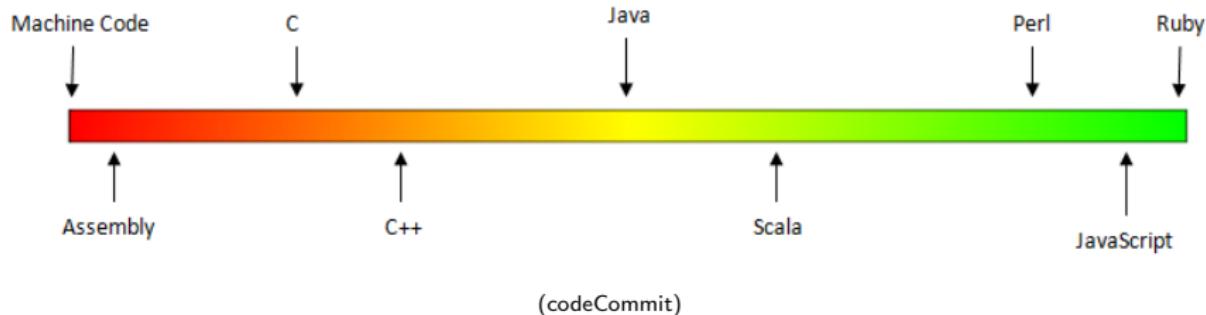
- Can view programming languages on a continuum.

Low-Level vs. High-Level Languages



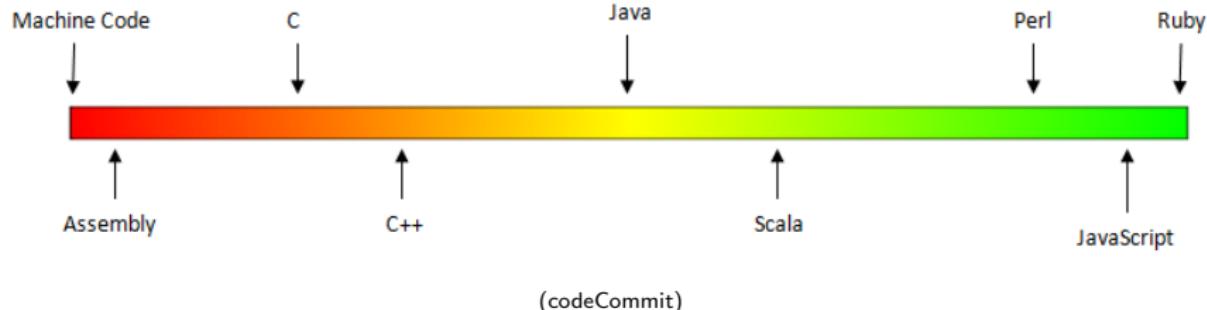
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages**

Low-Level vs. High-Level Languages



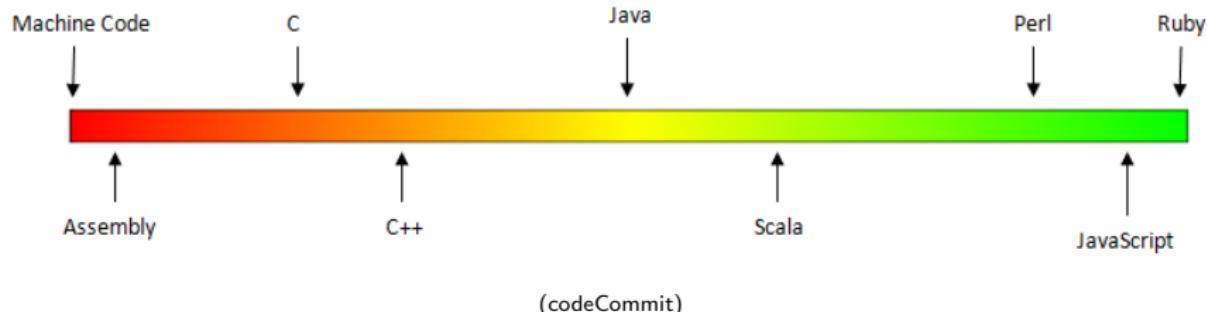
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).

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.

Processing

Bei [Lernzettel](#) für [Blindtext](#). "Blindtext" ist ein Dokument, das aussieht wie normaler Text, aber nur aus Ziffern besteht. Es kann nicht gelesen werden, ohne es zuvor ausdrucken oder es mit einem Blindenschriftleser zu öffnen.

Dies ist ein Blindtext. An ihm lässt sich vieles über die Schrift ableSEN, in der er gesetzt ist. Auf den ersten Blick wird der Grauwert der Schriftfläche sichtbar. Dann kann man prüfen, wie gut die Schrift zu lesen ist und wie sie auf den Leser wirkt.
Dies ist ein Blindtext. An ihm lässt sich

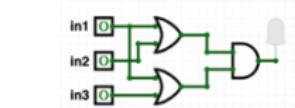


Data
&
Instructions

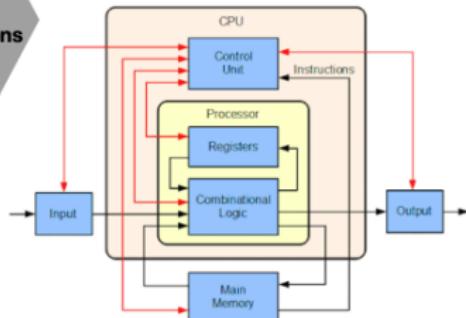
```
01110100011100100110000101  
1110010001101101110101011  
00111110001101000 101011  
00101110 00010000111111  
1110110111101000111011011  
001011010 101101 10011000  
01100101011010001000001  
11100100001101 00111101  
011000101101100011010101  
01000100000011000100 000  
0110010101 100110100101  
10010 1000000110110011101  
01101100110010000000000  
1110001000000000000000000  
011000110 10111010111001  
0000110000000000000000000  
011000110 10111010111001  
1001000000000000000000000  
00010 10010011 100100100  
111010000110111011011 01000  
1 1101101110011101001001  
0111001001011100 10001101  
00000110110001 10110 0011
```

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

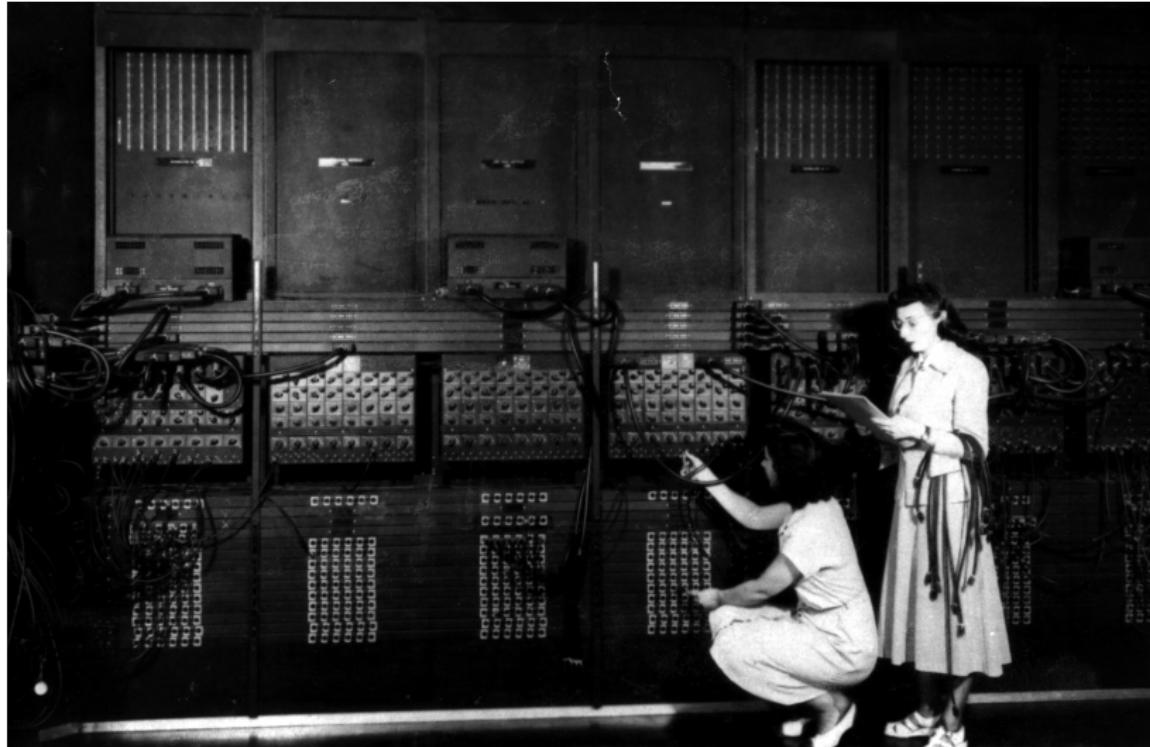
Data
&
Instructions



Circuits (switches)
On/Off 1/0 Logic
Billions of switches/bits



Machine Language



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

Machine Language

```
I FOX 12:01a 23- 1
A 002000 C2 30      REP #$30
A 002002 18          CLC
A 002003 F8          SED
A 002004 A9 34 12    LDA #$1234
A 002007 69 21 43    ADC #$4321
A 00200A 8F 03 7F 01 STA $017F03
A 00200E D8          CLD
A 00200F E2 30      SEP #$30
A 002011 00          BRK
A 2012

r
PB PC  NUMxDIZC .A .X .Y SP DP DB
; 00 E012 00110000 0000 0000 0002 CFFF 0000 00
g 2000

BREAK

PB PC  NUMxDIZC .A .X .Y SP DP DB
; 00 2013 00110000 5555 0000 0002 CFFF 0000 00
m 7f03 7f03
>007F03 55 55 00 00 00 00 00 00 00 00 00 00 00 00:UU .....
```

(wiki)

Machine Language

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

The screenshot shows a terminal window with two sections of text. The top section displays assembly code:

```
R 002000 C2 3B      RBF #438
R 002002 1B          CLC
R 002003 FB          SED
R 002004 69 34 12    LDA #1234
R 002007 69 21 43    ADC #4321
R 002008 8F 03 7F 01  STA #017F03
R 002009 00           CLD
R 00200F E2 30       SEI
R 002011 89           MRSB
R 002012             BRK
```

The bottom section shows the corresponding binary memory dump:

```
PC PC MMw012C .A .X .Y SP DP IR
: 00 E012 00100000 0000 0000 0002 CFFF 0000 00
& 2000
```

Below the binary dump is the word "BREAK".

(wiki)

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.

The screenshot shows a terminal window with two columns of text. The left column contains assembly-like instructions:

```
R 002000 C2 3B      LDR $R3, -17($R1)
R 002002 1B      BEQ $R3, $0, 11
R 002003 FB      SED
R 002004 00        CLC
R 002005 69 34 12  LDR $R4, #1234
R 002007 69 21 43  ADC $R4,$R3
R 002008 0F 03 7F 01  STA $R1, #17FB3
R 002009 00        CLD
R 00200F E2 30      SEP $R3
R 002011 00        SEI
R 002012 00        BRK
```

The right column shows the corresponding binary bytes:

```
    PB PC MNW0012C .A .X .Y SP DP R0
: 00 E012 00100000 0000 0000 0002 CFFF 0000 00
& 20000
```

Below this, there is a section labeled "BREAK" followed by more binary data:

```
PB PC MNW0012C .A .X .Y SP DP R0
: 00 E013 00100000 5555 0000 000C CFFF 0000 00
& 20000
```

At the bottom of the terminal window, there is a horizontal scrollbar.

(wiki)

Machine Language

```

A 002300 C2 3B REP #3B
A 002302 7F CLC
A 002303 FB SED
A 002304 34 12 ADD #1224
A 002307 69 21 43 ADC #4321
A 002308 8F B3 7F B1 STA $0017F9
A 00230E D0 CLD
A 00230F E2 3B SEP #3B
A 002311 90 BPK
A 002312

F
PB PC Mm0:012C A X Y SP DP IR
; 00 2013 00110800 0550 0000 0002 CFFF 0000 00
$ 2000

BREAK

PB PC Mm0:012C A X Y SP DP IR
; 00 2013 00110800 0550 0000 0002 CFFF 0000 00
$ 7793 7793

$0017F9 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

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

Machine Language



The screenshot shows a terminal window with assembly code and its corresponding binary output. The assembly code includes instructions like LDI, ADD, SUB, and MUL. The binary output consists of two columns of hex values.

Assembly Instruction	Binary Value
LDI R0, 1000	00000000 00000000
ADD R0, R1, R2	00000001 00000000
SUB R0, R1, R2	00000002 00000000
MUL R0, R1, R2	00000003 00000000
SWI	00000004 00000000

(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.
- 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

Line: 3 Go!

Show/Hide Demos

User Guide | Unit Tests | Docs

Addition Doubler Stav Looper Stack Test Hello World

Code Gen Save String Interactive Binary2 Decimal Decimal2 Binary

Debug

```
1 # Store 'Hello world!' at the top of the stack
2 ADDI $sp, $sp, -13
3 ADDI $t0, $zero, 72 # H
4 SB $t0, 0($sp)
5 ADDI $t0, $zero, 101 # e
6 SB $t0, 1($sp)
7 ADDI $t0, $zero, 108 # l
8 SB $t0, 2($sp)
9 ADDI $t0, $zero, 108 # i
10 SB $t0, 3($sp)
11 ADDI $t0, $zero, 111 # o
12 SB $t0, 4($sp)
13 ADDI $t0, $zero, 32 # (space)
14 SB $t0, 5($sp)
15 ADDI $t0, $zero, 119 # w
16 SB $t0, 6($sp)
17 ADDI $t0, $zero, 111 # o
18 SB $t0, 7($sp)
19 ADDI $t0, $zero, 114 # r
20 SB $t0, 8($sp)
21 ADDI $t0, $zero, 108 # l
22 SB $t0, 9($sp)
23 ADDI $t0, $zero, 100 # d
24 SB $t0, 10($sp)
25 ADDI $t0, $zero, 33 # !
26 SB $t0, 11($sp)
27 ADDI $t0, $zero, 0 # (null)
28 SB $t0, 12($sp)
29
30 ADDI $v0, $zero, 4 # 4 is for print string
31 ADDI $a0, $sp, 0
32 syscall           # print to the log
```

Step	Run	<input checked="" type="checkbox"/> Enable auto switching			
S	T	A	V	Stack	Log
s0:				10	
s1:				9	
s2:				9	
s3:				22	
s4:				696	
s5:				976	
s6:				927	
s7:				418	

(WeMIPS)

WeMIPS

User | 3 | Dat | ShowWide Device | User Guide | Unit Tests | Docs

Addition Doubler | Stax | Looper | Stack Test | Hello World | Code Gen Save String | Interactive | Binary2 Decimal | Decimal2 Binary | Debug

```
# Store 'Hello world!' at the top of the stack
1    ADDI $t0,$zero,72 # $N
2    ADDI $t1,$zero,101 # $e
3    ADDI $t2,$zero,101 # $e
4    ADDI $t3,$zero,101 # $e
5    ADDI $t4,$zero,101 # $e
6    ADDI $t5,$zero,101 # $e
7    ADDI $t6,$zero,101 # $e
8    ADDI $t7,$zero,101 # $e
9    ADDI $t8,$zero,101 # $e
10   ADDI $t9,$zero,101 # $e
11   ADDI $t10,$zero,101 # $e
12   ADDI $t11,$zero,41#$aay
13   ADDI $t12,$zero,33 # $(apex)
14   ADDI $t13,$zero,51#$bby
15   ADDI $t14,$zero,119 # $(null)
16   ADDI $t15,$zero,119 # $(null)
17   ADDI $t16,$zero,119 # $(null)
18   ADDI $t17,$zero,119 # $(null)
19   ADDI $t18,$zero,119 # $(null)
20   ADDI $t19,$zero,119 # $(null)
21   ADDI $t20,$zero,119 # $(null)
22   ADDI $t21,$zero,119 # $(null)
23   ADDI $t22,$zero,119 # $(null)
24   ADDI $t23,$zero,119 # $(null)
25   ADDI $t24,$zero,109 # $d
26   ADDI $t25,$zero,109 # $d
27   ADDI $t26,$zero,33 # $i
28   ADDI $t27,$zero,119 # $(null)
29   ADDI $t28,$zero,0 # $(null)
30   ADDI $t29,$zero,119 # $(null)
31   ADDI $t30,$zero,4 # 4 is for print string
32   ADDI $t31,$zero,0 # point to the log
33   syscall
```

Step	Run	<input checked="" type="checkbox"/> Enable auto switching			
S	T	A	V	Stack	Log
s0:	10				
s1:	9				
s2:	8				
s3:	22				
s4:	695				
s5:	976				
s6:	977				
s7:	419				

(Demo with WeMIPS)

MIPS Commands



The screenshot shows a MIPS assembly debugger interface. At the top, there's a menu bar with "File", "Edit", "Get", "Show/Hide Demo", "User Guide | Unit Tests | Docs", and tabs for "Assembly", "Assembler", "Run", "Looper", "Stack View", "Hello World", "Code Gen", "Save String", "Interactive", "Decimal", "Decimal/Hex", and "Decimal/Binary". Below the menu is a toolbar with "Step", "Run", "Break", and "Create auto watching". A status bar at the bottom shows assembly, binary, and hex values.

The assembly code is as follows:

```
1 # Ensure 'Hello, world!' is at the top of the stack
2 .text
3 .globl _start
4 .type _start, @function
5 _start:
6    li $t0, 111990
7    sb $t0, $19($sp)
8    li $t0, 111991
9    sb $t0, $19($sp)
10   li $t0, 111992
11   sb $t0, $19($sp)
12   li $t0, 111993
13   sb $t0, $19($sp)
14   li $t0, 111994
15   sb $t0, $19($sp)
16   li $t0, 111995
17   sb $t0, $19($sp)
18   li $t0, 111996
19   sb $t0, $19($sp)
20   li $t0, 111997
21   sb $t0, $19($sp)
22   li $t0, 111998
23   sb $t0, $19($sp)
24   li $t0, 101999
25   sb $t0, $19($sp)
26   li $t0, 111999
27   sb $t0, $19($sp)
28   li $t0, 0
29   sb $t0, $19($sp)
30   li $t0, 4 # $t0 is the print string
31   addi $t0, $sp, 4 # point to the log
32   syscall
```

The registers shown in the status bar are:

S	T	A	V	Stack	Log
x0	10				
x1	9				
x2	8				
x3	22				
x4	000				
x5	819				
x6	827				
x7	411				

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

MIPS Commands

The screenshot shows the QEMU debugger interface. At the top, there are tabs for "Show/Hide Demo", "User Guide", "Unit Tests", and "Docs". Below that is a menu bar with "File", "Edit", "Run", "Help", "Registers", "Stack View", "Hello World", "Code Gen Base String", "Interactive", "Decimal Decimal", and "Decimal Binary". A "Debug" button is also present.

The assembly code window displays the following MIPS assembly code:

```
1 # Ensure '$main_stack' is at the top of the stack
2
3 .data
4 .text
5
6 li $t0, $main_stack; $t0 = &main_stack
7 sb $t0, $11($sp); $t0 = main_stack[1]
8
9 li $t1, 1111001; $t1 = 1111001
10 sb $t1, $11($sp); $t1 = 1111001
11
12 li $t2, 1110001; $t2 = 1110001
13 sb $t2, $11($sp); $t2 = 1110001
14
15 li $t3, 1110000; $t3 = 1110000
16 sb $t3, $11($sp); $t3 = 1110000
17
18 add $t4, $t0, $t1, $t2 # (expand)
19 add $t5, $t0, $t2, $t3 # (expand)
20 add $t6, $t0, $t3, $t4 # (expand)
21 add $t7, $t0, $t4, $t5 # (expand)
22 add $t8, $t0, $t5, $t6 # (expand)
23 add $t9, $t0, $t6, $t7 # (expand)
24 add $t10, $t0, $t7, $t8 # (expand)
25 add $t11, $t0, $t8, $t9 # (expand)
26 add $t12, $t0, $t9, $t10 # (expand)
27 add $t13, $t0, $t10, $t11 # (expand)
28 add $t14, $t0, $t11, $t12 # (expand)
29 add $t15, $t0, $t12, $t13 # (expand)
30 add $t16, $t0, $t13, $t14 # (expand)
31 add $t17, $t0, $t14, $t15 # (expand)
32 add $t18, $t0, $t15, $t16 # (expand)
33 add $t19, $t0, $t16, $t17 # (expand)
34 add $t20, $t0, $t17, $t18 # (expand)
35 add $t21, $t0, $t18, $t19 # (expand)
36 add $t22, $t0, $t19, $t20 # (expand)
37 add $t23, $t0, $t20, $t21 # (expand)
38 add $t24, $t0, $t21, $t22 # (expand)
39 add $t25, $t0, $t22, $t23 # (expand)
40 add $t26, $t0, $t23, $t24 # (expand)
41 add $t27, $t0, $t24, $t25 # (expand)
42 add $t28, $t0, $t25, $t26 # (expand)
43 add $t29, $t0, $t26, $t27 # (expand)
44 add $t30, $t0, $t27, $t28 # (expand)
45 add $t31, $t0, $t28, $t29 # (expand)
46 add $t32, $t0, $t29, $t30 # (expand)
47 add $t33, $t0, $t30, $t31 # (expand)
48 add $t34, $t0, $t31, $t32 # (expand)
49 add $t35, $t0, $t32, $t33 # (expand)
50 add $t36, $t0, $t33, $t34 # (expand)
51 add $t37, $t0, $t34, $t35 # (expand)
52 add $t38, $t0, $t35, $t36 # (expand)
53 add $t39, $t0, $t36, $t37 # (expand)
54 add $t40, $t0, $t37, $t38 # (expand)
55 add $t41, $t0, $t38, $t39 # (expand)
56 add $t42, $t0, $t39, $t40 # (expand)
57 add $t43, $t0, $t40, $t41 # (expand)
58 add $t44, $t0, $t41, $t42 # (expand)
59 add $t45, $t0, $t42, $t43 # (expand)
60 add $t46, $t0, $t43, $t44 # (expand)
61 add $t47, $t0, $t44, $t45 # (expand)
62 add $t48, $t0, $t45, $t46 # (expand)
63 add $t49, $t0, $t46, $t47 # (expand)
64 add $t50, $t0, $t47, $t48 # (expand)
65 add $t51, $t0, $t48, $t49 # (expand)
66 add $t52, $t0, $t49, $t50 # (expand)
67 add $t53, $t0, $t50, $t51 # (expand)
68 add $t54, $t0, $t51, $t52 # (expand)
69 add $t55, $t0, $t52, $t53 # (expand)
70 add $t56, $t0, $t53, $t54 # (expand)
71 add $t57, $t0, $t54, $t55 # (expand)
72 add $t58, $t0, $t55, $t56 # (expand)
73 add $t59, $t0, $t56, $t57 # (expand)
74 add $t60, $t0, $t57, $t58 # (expand)
75 add $t61, $t0, $t58, $t59 # (expand)
76 add $t62, $t0, $t59, $t60 # (expand)
77 add $t63, $t0, $t60, $t61 # (expand)
78 add $t64, $t0, $t61, $65279 # ($t0 = 65279)
```

The register window shows the following values:

S	T	V	Stack	Log
x0	10			
x1	9			
x2	8			
x3	22			
x4	0000			
x5	819			
x6	827			
x7	411			

- **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

The screenshot shows a MIPS assembly debugger interface. The code window displays assembly instructions for a program. The register window on the right shows the values of registers \$0 through \$7. Below the register window are control buttons for Step, Run, Break, and Stop.

Register	Value
\$0	10
\$1	9
\$2	8
\$3	22
\$4	000
\$5	879
\$6	827
\$7	411

- **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

The screenshot shows a MIPS assembly debugger interface. At the top, there's a menu bar with 'File', 'Edit', 'Get', 'Show/Hide Demo', 'User Guide | Unit Tests | Docs', and tabs for 'Assembly', 'Data', 'Hex', 'Looper', 'Stack View', 'Hello World', 'Code Gen', 'Save String', 'Interactive', 'Decimal Decimal', 'Decimal Binary', and 'Debug'. Below the menu is a code editor window containing the following assembly code:

```
1 # Shows 'Hello world' at the top of the stack
2
3 .text
4 .globl _start
5
6 .data
7 _str: .asciz "Hello world\n"
8
9 .text
10 _start:
11     li $t0, _str
12     li $t1, 11
13     add $t2, $t0, $t1
14     li $v0, 4
15     la $a0, _str
16     syscall
17
18     li $t0, _str
19     li $t1, 11
20     add $t2, $t0, $t1
21     li $v0, 4
22     la $a0, _str
23     syscall
24
25     li $t0, 10
26     li $t1, 10
27     add $t2, $t0, $t1
28     li $v0, 11
29     la $a0, _str
30     syscall
31
32     li $t0, 10
33     li $t1, 10
34     add $t2, $t0, $t1
35     li $v0, 11
36     la $a0, _str
37     syscall
38
39     li $t0, 10
40     li $t1, 10
41     add $t2, $t0, $t1
42     li $v0, 11
43     la $a0, _str
44     syscall
45
46     li $t0, 10
47     li $t1, 10
48     add $t2, $t0, $t1
49     li $v0, 11
50     la $a0, _str
51     syscall
52
53     li $t0, 10
54     li $t1, 10
55     add $t2, $t0, $t1
56     li $v0, 11
57     la $a0, _str
58     syscall
59
60     li $t0, 10
61     li $t1, 10
62     add $t2, $t0, $t1
63     li $v0, 11
64     la $a0, _str
65     syscall
66
67     li $t0, 10
68     li $t1, 10
69     add $t2, $t0, $t1
70     li $v0, 11
71     la $a0, _str
72     syscall
73
74     li $t0, 10
75     li $t1, 10
76     add $t2, $t0, $t1
77     li $v0, 11
78     la $a0, _str
79     syscall
80
81     li $t0, 10
82     li $t1, 10
83     add $t2, $t0, $t1
84     li $v0, 11
85     la $a0, _str
86     syscall
87
88     li $t0, 10
89     li $t1, 10
90     add $t2, $t0, $t1
91     li $v0, 11
92     la $a0, _str
93     syscall
94
95     li $t0, 10
96     li $t1, 10
97     add $t2, $t0, $t1
98     li $v0, 11
99     la $a0, _str
100    syscall
101
102    li $t0, 10
103    li $t1, 10
104    add $t2, $t0, $t1
105    li $v0, 11
106    la $a0, _str
107    syscall
108
109    li $t0, 10
110    li $t1, 10
111    add $t2, $t0, $t1
112    li $v0, 11
113    la $a0, _str
114    syscall
115
116    li $t0, 10
117    li $t1, 10
118    add $t2, $t0, $t1
119    li $v0, 11
120    la $a0, _str
121    syscall
122
123    li $t0, 10
124    li $t1, 10
125    add $t2, $t0, $t1
126    li $v0, 11
127    la $a0, _str
128    syscall
129
130    li $t0, 10
131    li $t1, 10
132    add $t2, $t0, $t1
133    li $v0, 11
134    la $a0, _str
135    syscall
136
137    li $t0, 10
138    li $t1, 10
139    add $t2, $t0, $t1
140    li $v0, 11
141    la $a0, _str
142    syscall
143
144    li $t0, 10
145    li $t1, 10
146    add $t2, $t0, $t1
147    li $v0, 11
148    la $a0, _str
149    syscall
150
151    li $t0, 10
152    li $t1, 10
153    add $t2, $t0, $t1
154    li $v0, 11
155    la $a0, _str
156    syscall
157
158    li $t0, 10
159    li $t1, 10
160    add $t2, $t0, $t1
161    li $v0, 11
162    la $a0, _str
163    syscall
164
165    li $t0, 10
166    li $t1, 10
167    add $t2, $t0, $t1
168    li $v0, 11
169    la $a0, _str
170    syscall
171
172    li $t0, 10
173    li $t1, 10
174    add $t2, $t0, $t1
175    li $v0, 11
176    la $a0, _str
177    syscall
178
179    li $t0, 10
180    li $t1, 10
181    add $t2, $t0, $t1
182    li $v0, 11
183    la $a0, _str
184    syscall
185
186    li $t0, 10
187    li $t1, 10
188    add $t2, $t0, $t1
189    li $v0, 11
190    la $a0, _str
191    syscall
192
193    li $t0, 10
194    li $t1, 10
195    add $t2, $t0, $t1
196    li $v0, 11
197    la $a0, _str
198    syscall
199
200    li $t0, 10
201    li $t1, 10
202    add $t2, $t0, $t1
203    li $v0, 11
204    la $a0, _str
205    syscall
206
207    li $t0, 10
208    li $t1, 10
209    add $t2, $t0, $t1
210    li $v0, 11
211    la $a0, _str
212    syscall
213
214    li $t0, 10
215    li $t1, 10
216    add $t2, $t0, $t1
217    li $v0, 11
218    la $a0, _str
219    syscall
220
221    li $t0, 10
222    li $t1, 10
223    add $t2, $t0, $t1
224    li $v0, 11
225    la $a0, _str
226    syscall
227
228    li $t0, 10
229    li $t1, 10
230    add $t2, $t0, $t1
231    li $v0, 11
232    la $a0, _str
233    syscall
234
235    li $t0, 10
236    li $t1, 10
237    add $t2, $t0, $t1
238    li $v0, 11
239    la $a0, _str
240    syscall
241
242    li $t0, 10
243    li $t1, 10
244    add $t2, $t0, $t1
245    li $v0, 11
246    la $a0, _str
247    syscall
248
249    li $t0, 10
250    li $t1, 10
251    add $t2, $t0, $t1
252    li $v0, 11
253    la $a0, _str
254    syscall
255
256    li $t0, 10
257    li $t1, 10
258    add $t2, $t0, $t1
259    li $v0, 11
260    la $a0, _str
261    syscall
262
263    li $t0, 10
264    li $t1, 10
265    add $t2, $t0, $t1
266    li $v0, 11
267    la $a0, _str
268    syscall
269
270    li $t0, 10
271    li $t1, 10
272    add $t2, $t0, $t1
273    li $v0, 11
274    la $a0, _str
275    syscall
276
277    li $t0, 10
278    li $t1, 10
279    add $t2, $t0, $t1
280    li $v0, 11
281    la $a0, _str
282    syscall
283
284    li $t0, 10
285    li $t1, 10
286    add $t2, $t0, $t1
287    li $v0, 11
288    la $a0, _str
289    syscall
290
291    li $t0, 10
292    li $t1, 10
293    add $t2, $t0, $t1
294    li $v0, 11
295    la $a0, _str
296    syscall
297
298    li $t0, 10
299    li $t1, 10
300    add $t2, $t0, $t1
301    li $v0, 11
302    la $a0, _str
303    syscall
304
305    li $t0, 10
306    li $t1, 10
307    add $t2, $t0, $t1
308    li $v0, 11
309    la $a0, _str
310    syscall
311
312    li $t0, 10
313    li $t1, 10
314    add $t2, $t0, $t1
315    li $v0, 11
316    la $a0, _str
317    syscall
318
319    li $t0, 10
320    li $t1, 10
321    add $t2, $t0, $t1
322    li $v0, 11
323    la $a0, _str
324    syscall
325
326    li $t0, 10
327    li $t1, 10
328    add $t2, $t0, $t1
329    li $v0, 11
330    la $a0, _str
331    syscall
332
333    li $t0, 10
334    li $t1, 10
335    add $t2, $t0, $t1
336    li $v0, 11
337    la $a0, _str
338    syscall
339
340    li $t0, 10
341    li $t1, 10
342    add $t2, $t0, $t1
343    li $v0, 11
344    la $a0, _str
345    syscall
346
347    li $t0, 10
348    li $t1, 10
349    add $t2, $t0, $t1
350    li $v0, 11
351    la $a0, _str
352    syscall
353
354    li $t0, 10
355    li $t1, 10
356    add $t2, $t0, $t1
357    li $v0, 11
358    la $a0, _str
359    syscall
360
361    li $t0, 10
362    li $t1, 10
363    add $t2, $t0, $t1
364    li $v0, 11
365    la $a0, _str
366    syscall
367
368    li $t0, 10
369    li $t1, 10
370    add $t2, $t0, $t1
371    li $v0, 11
372    la $a0, _str
373    syscall
374
375    li $t0, 10
376    li $t1, 10
377    add $t2, $t0, $t1
378    li $v0, 11
379    la $a0, _str
380    syscall
381
382    li $t0, 10
383    li $t1, 10
384    add $t2, $t0, $t1
385    li $v0, 11
386    la $a0, _str
387    syscall
388
389    li $t0, 10
390    li $t1, 10
391    add $t2, $t0, $t1
392    li $v0, 11
393    la $a0, _str
394    syscall
395
396    li $t0, 10
397    li $t1, 10
398    add $t2, $t0, $t1
399    li $v0, 11
400    la $a0, _str
401    syscall
402
403    li $t0, 10
404    li $t1, 10
405    add $t2, $t0, $t1
406    li $v0, 11
407    la $a0, _str
408    syscall
409
410    li $t0, 10
411    li $t1, 10
412    add $t2, $t0, $t1
413    li $v0, 11
414    la $a0, _str
415    syscall
416
417    li $t0, 10
418    li $t1, 10
419    add $t2, $t0, $t1
420    li $v0, 11
421    la $a0, _str
422    syscall
423
424    li $t0, 10
425    li $t1, 10
426    add $t2, $t0, $t1
427    li $v0, 11
428    la $a0, _str
429    syscall
430
431    li $t0, 10
432    li $t1, 10
433    add $t2, $t0, $t1
434    li $v0, 11
435    la $a0, _str
436    syscall
437
438    li $t0, 10
439    li $t1, 10
440    add $t2, $t0, $t1
441    li $v0, 11
442    la $a0, _str
443    syscall
444
445    li $t0, 10
446    li $t1, 10
447    add $t2, $t0, $t1
448    li $v0, 11
449    la $a0, _str
450    syscall
451
452    li $t0, 10
453    li $t1, 10
454    add $t2, $t0, $t1
455    li $v0, 11
456    la $a0, _str
457    syscall
458
459    li $t0, 10
460    li $t1, 10
461    add $t2, $t0, $t1
462    li $v0, 11
463    la $a0, _str
464    syscall
465
466    li $t0, 10
467    li $t1, 10
468    add $t2, $t0, $t1
469    li $v0, 11
470    la $a0, _str
471    syscall
472
473    li $t0, 10
474    li $t1, 10
475    add $t2, $t0, $t1
476    li $v0, 11
477    la $a0, _str
478    syscall
479
480    li $t0, 10
481    li $t1, 10
482    add $t2, $t0, $t1
483    li $v0, 11
484    la $a0, _str
485    syscall
486
487    li $t0, 10
488    li $t1, 10
489    add $t2, $t0, $t1
490    li $v0, 11
491    la $a0, _str
492    syscall
493
494    li $t0, 10
495    li $t1, 10
496    add $t2, $t0, $t1
497    li $v0, 11
498    la $a0, _str
499    syscall
499
```

Below the code editor is a table showing register values:

S	T	A	V	Stack	Log
x0	10				
x1	9				
x2	8				
x3	22				
x4	0000				
x5	819				
x6	827				
x7	411				

- **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

MIPS Commands

The screenshot shows a MIPS assembly editor interface. At the top, there are tabs for "Show/Hide Demo", "User Guide", "Unit Tests", and "Docs". Below the tabs are buttons for "Addition", "Subtraction", "Mul", "Division", "Stack Test", and "Hello World". There are also buttons for "Code Gen", "Save String", "Interactive", "Decimal Decimal", and "Decimal Binary". A "Debug" button is also present.

The main area contains assembly code:

```
1 # Shows 'Hello world' at the top of the stack
2 .text
3 .globl _start
4 .type _start, @function
5 _start:
6    li $t0, 11110000
7    li $t1, 11100000
8    li $t2, 11100001
9    li $t3, 11100010
10   li $t4, 11100011
11   li $t5, 11100100
12   li $t6, 11100101
13   li $t7, 11100110, 32 # ($apsp)
14   addi $t8, $t6, $t7
15   addi $t9, $t8, $t7
16   addi $t10, $t9, $t7
17   addi $t11, $t10, $t7
18   addi $t12, $t11, $t7
19   addi $t13, $t12, $t7
20   addi $t14, $t13, $t7
21   addi $t15, $t14, $t7
22   li $t16, 11100111
23   li $t17, 11101000
24   li $t18, 10100000, 100 # $
25   li $t19, 10100001
26   li $t20, 11101001
27   li $t21, 0 # ($at)
28   li $t22, 11101010
29   addi $t23, $t21, $t22
30   addi $t24, $t23, $t22
31   # $t24 = 11101011, 4 # $ is for print string
32   syscall
```

To the right of the code, there is a register table with columns for S, T, A, V, Stack, and Log. The registers show the following values:

S	T	A	V	Stack	Log
x0				10	
x1				9	
x2				8	
x3				22	
x4				0000	
x5				879	
x6				827	
x7				411	

- **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.

MIPS Commands

```
Line 3 Set Showchee Demos User Guide | List Tasks | Close
Additional Decoder Hex Loader Stack Test Help Window
Code Gen Save String Interactive Binary Decimal Decimal/Hexadecimal
Debug

d Shows the local code at the top of the stack
d0 00401000 00401000 $0000 0000000000000000
d1 00401000 00401000 $0000 0000000000000000
d2 00401000 00401000 $0000 0000000000000000
d3 00401000 00401000 $0000 0000000000000000
d4 00401000 00401000 $0000 0000000000000000
d5 00401000 00401000 $0000 0000000000000000
d6 00401000 00401000 $0000 0000000000000000
d7 00401000 00401000 $0000 0000000000000000
d8 00401000 00401000 $0000 0000000000000000
d9 00401000 00401000 $0000 0000000000000000
dA 00401000 00401000 $0000 0000000000000000
dB 00401000 00401000 $0000 0000000000000000
dC 00401000 00401000 $0000 0000000000000000
dD 00401000 00401000 $0000 0000000000000000
dE 00401000 00401000 $0000 0000000000000000
dF 00401000 00401000 $0000 0000000000000000
Step Run ✓ Enable auto switching
S T A V Stack Log

d0 10
d1 9
d2 8
d3 20
d4 656
d5 676
d6 807
d7 418

d Shows the local code at the top of the stack
d0 00401000 00401000 $0000 0000000000000000
d1 00401000 00401000 $0000 0000000000000000
d2 00401000 00401000 $0000 0000000000000000
d3 00401000 00401000 $0000 0000000000000000
d4 00401000 00401000 $0000 0000000000000000
d5 00401000 00401000 $0000 0000000000000000
d6 00401000 00401000 $0000 0000000000000000
d7 00401000 00401000 $0000 0000000000000000
d8 00401000 00401000 $0000 0000000000000000
d9 00401000 00401000 $0000 0000000000000000
dA 00401000 00401000 $0000 0000000000000000
dB 00401000 00401000 $0000 0000000000000000
dC 00401000 00401000 $0000 0000000000000000
dD 00401000 00401000 $0000 0000000000000000
dE 00401000 00401000 $0000 0000000000000000
dF 00401000 00401000 $0000 0000000000000000
ADDQ $10, %esp;    # esp = 0 for prior saving
movl %esp, %ebp;   # points to the top
syscall;
```

- **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

The screenshot shows a MIPS assembly editor interface. At the top, there's a menu bar with 'File', 'Edit', 'Run', 'Help', 'Show/Hide Demo', and tabs for 'Assembly', 'Hex', 'Looper', 'Stack View', 'Hello World', 'Code Gen', 'Data String', 'Interactive', 'Binary', 'Decimal', 'Hexadecimal', and 'Decimal/Hex'. Below the menu is a toolbar with icons for 'Step', 'Run', 'Break', 'Stop', and 'Log'. On the right, there's a register window titled 'Registers' with columns for Register Name (x0-x7) and Value. The values are: x0=10, x1=9, x2=8, x3=22, x4=000, x5=819, x6=927, x7=411. The central area contains assembly code:

```
# Assume 'Hello, world!' at the top of the stack
        .text
        .globl _start
_start:    li $t0, 11110000
        sb $t0, 11110000($sp)
        li $t1, 11110001
        sb $t1, 11110001($sp)
        li $t2, 11110002
        sb $t2, 11110002($sp)
        li $t3, 11110003
        sb $t3, 11110003($sp)
        li $t4, 11110004
        sb $t4, 11110004($sp)
        li $t5, 11110005
        sb $t5, 11110005($sp)
        li $t6, 11110006
        sb $t6, 11110006($sp)
        li $t7, 11110007
        sb $t7, 11110007($sp)
        addi $t0, $t0, 1111 # (pop)
        addi $t1, $t1, 1111 # (pop)
        addi $t2, $t2, 1111 # (pop)
        addi $t3, $t3, 1111 # (pop)
        addi $t4, $t4, 1111 # (pop)
        addi $t5, $t5, 1111 # (pop)
        addi $t6, $t6, 1111 # (pop)
        addi $t7, $t7, 1111 # (pop)
        addi $t0, $t0, 1111 # for print string
        addi $t0, $t0, 0 # (null)
        addi $t0, $t0, 1111 # point to the log
        syscall
```

- **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 (Basic form: OP label)

Challenge:

Line: 3 Go! Show/Hide Demos

User Guide | Unit Tests | Docs

Addition Doubler Stav Looper Stack Test Hello World

Code Gen Save String Interactive Binary2 Decimal Decimal2 Binary

Debug

```
1 # Store 'Hello world!' at the top of the stack
2 ADDI $sp, $sp, -13
3 ADDI $t0, $zero, 72 # H
4 SB $t0, 0($sp)
5 ADDI $t0, $zero, 101 # e
6 SB $t0, 1($sp)
7 ADDI $t0, $zero, 108 # l
8 SB $t0, 2($sp)
9 ADDI $t0, $zero, 108 # l
10 SB $t0, 3($sp)
11 ADDI $t0, $zero, 111 # o
12 SB $t0, 4($sp)
13 ADDI $t0, $zero, 32 # (space)
14 SB $t0, 5($sp)
15 ADDI $t0, $zero, 119 # w
16 SB $t0, 6($sp)
17 ADDI $t0, $zero, 111 # o
18 SB $t0, 7($sp)
19 ADDI $t0, $zero, 114 # r
20 SB $t0, 8($sp)
21 ADDI $t0, $zero, 108 # l
22 SB $t0, 9($sp)
23 ADDI $t0, $zero, 100 # d
24 SB $t0, 10($sp)
25 ADDI $t0, $zero, 33 # !
26 SB $t0, 11($sp)
27 ADDI $t0, $zero, 0 # (null)
28 SB $t0, 12($sp)
29
30 ADDI $v0, $zero, 4 # 4 is for print string
31 ADDI $a0, $sp, 0      # print to the log
32 syscall
```

Step Run Enable auto switching

S	T	A	V	Stack	Log
s0:	10				
s1:	9				
s2:	9				
s3:	22				
s4:	696				
s5:	976				
s6:	927				
s7:	418				

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

WeMIPS

The screenshot shows the WeMIPS IDE interface. At the top, there are tabs for 'Run', '32bit', '64bit', and 'ShowWide Device'. Below these are navigation links: 'Addition Doubler', 'Stax', 'Looper', 'Stack Test', and 'Hello World'. Underneath are buttons for 'Code Gen Save String', 'Interactive', 'Binary2 Decimal', and 'Decimal2 Binary'. A 'Debug' button is also present.

The main area displays assembly code:

```
# Store 'Hello world!' at the top of the stack
1    ADDI $t0,$zero,72 # N
2    ADDI $t1,$zero,101 # e
3    ADDI $t2,$zero,101 # m
4    ADDI $t3,$zero,101 # l
5    ADDI $t4,$zero,101 # o
6    ADDI $t5,$zero,101 # r
7    ADDI $t6,$zero,101 # i
8    ADDI $t7,$zero,101 # n
9    ADDI $t8,$zero,101 # d
10   ADDI $t9,$zero,33 # !
11   ADDI $t10,$zero,0 # null
12   ADDI $t11,$zero,4 # 4 is for print string
13   ADDI $t12,$zero,0 # point to the log
14
15   #syscall
```

To the right of the assembly code is a debugger interface with tabs for 'Step', 'Run' (which is selected), and 'Log'. The 'Log' tab shows the following register values:

S	T	A	V	Stack	Log
s0	10				
s1	9				
s2	8				
s3	22				
s4	695				
s5	976				
s6	977				
s7	419				

(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.



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:



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`.



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.



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

Line:	1	Go!
-------	---	-----

Show/Hide Demos

Addition Doubler	Stav	Looper	Stack Test	Hello World	Code Gen Save String	Interactive
Binary2 Decimal	Decimal2 Binary	Debug				

[User Guide](#) | [Unit Tests](#) | [Docs](#)

```
1 ADDI $sp, $sp, -27      # Set up stack
2 ADDI $s3, $zero, 1       # Store 1 in a register
3 ADDI $t0, $zero, 97      # Set $t0 at 97 (a)
4 ADDI $s2, $zero, 26      # Use to loop 26 times
5 SETUP: SB $t0, 0($sp)    # Next letter in $t0
6 ADDI $sp, $sp, 1         # Increment the stack
7 SUB $s2, $s2, $s3        # Decrease the counter by 1
8 ADDI $t0, $t0, 1         # Increment the letter
9 BEQ $s2, $zero, DONE     # Jump to done if $s2 == 0
10 J SETUP                 # Else, jump back to SETUP
11 DONE: ADDI $t0, $zero, 0 # Null (0) to terminate string
12 SB $t0, 0($sp)          # Add null to stack
13 ADDI $sp, $sp, -27      # Set up stack to print
14
15 ADDI $v0, $zero, 4       # 4 is for print string
16 ADDI $a0, $sp, 0         # Set $a0 to stack pointer
17 syscall                 # Print to the log
```

(Demo
with
WeMIPS)

Enable auto switching

S T A V Stack Log

Emulation complete, returning to line 1

abcdefghijklmnopqrstuvwxyz

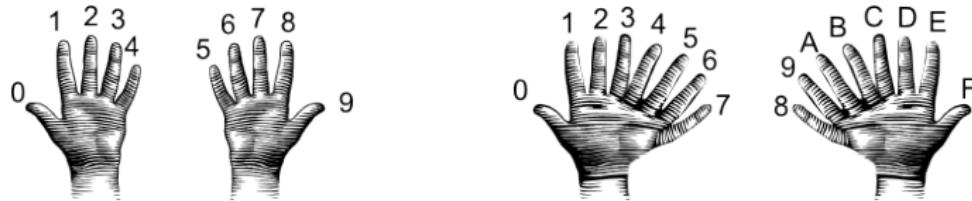


Today's Topics



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

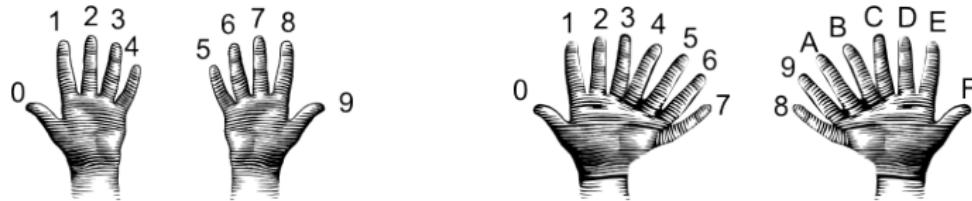
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
 - Convert first digit to decimal and multiple by 16.

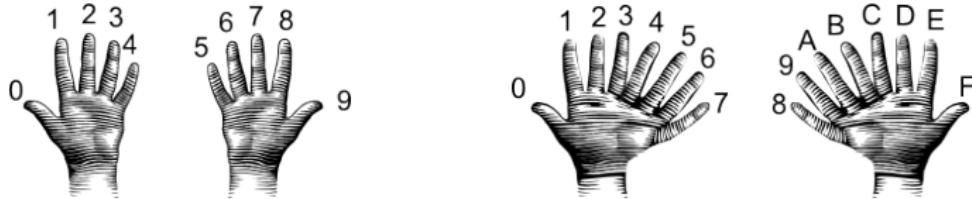
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
 - Convert first digit to decimal and multiple by 16.
 - Convert second digit to decimal and add to total.

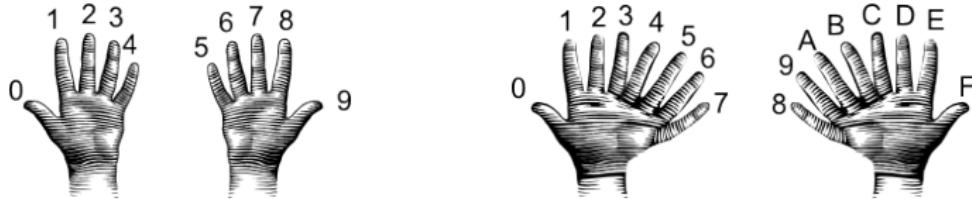
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
 - 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?

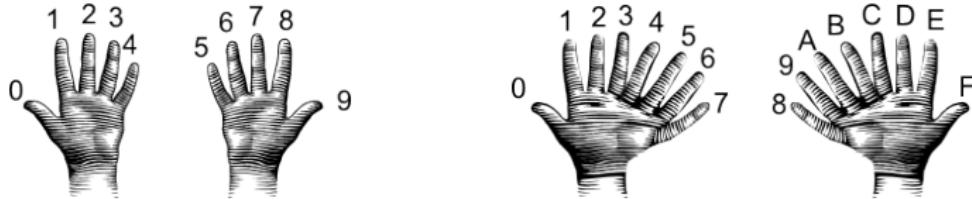
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
 - 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.

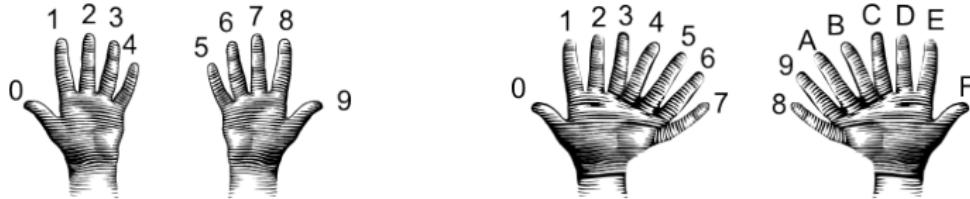
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
 - 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.

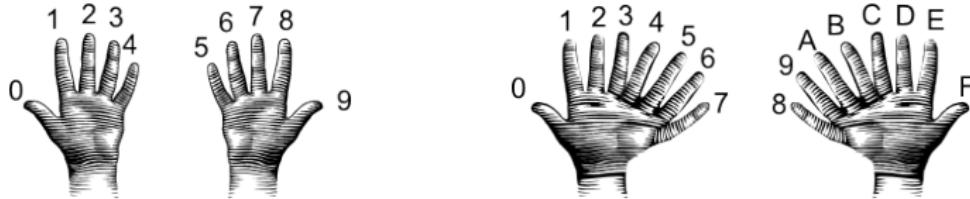
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
 - 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.

Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):

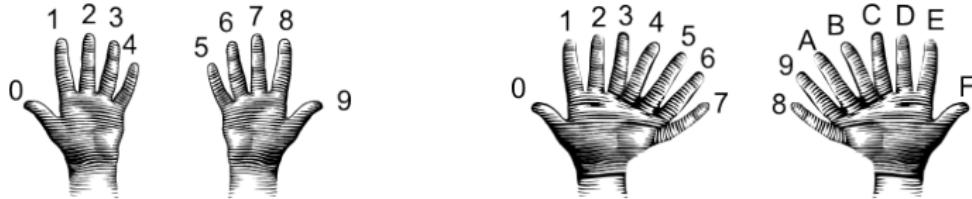
- Convert first digit to decimal and multiple by 16.
 - Convert second digit to decimal and add to total.
 - Example: what is 2A as a decimal number?

2 in decimal is 2. 2×16 is 32.

A in decimal digits is 10.

$32 + 10$ is 42.

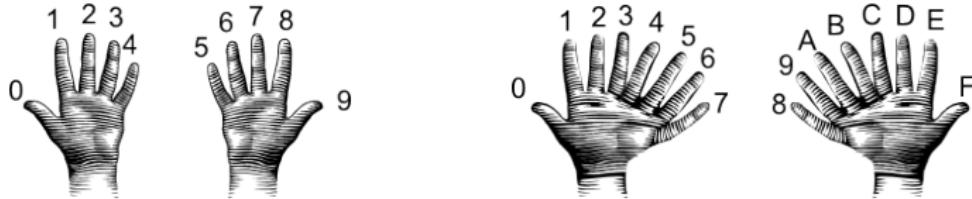
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
 - 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?

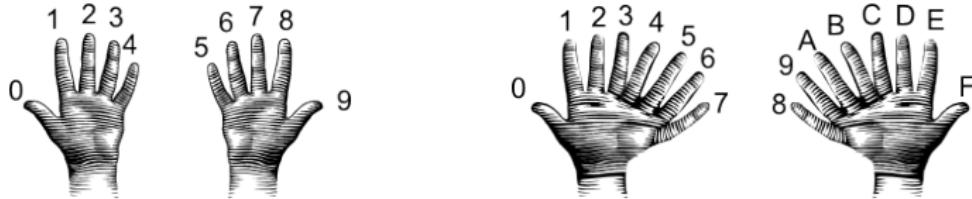
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
 - 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.

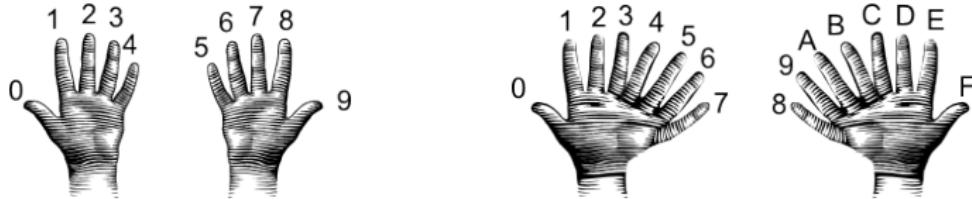
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
 - 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.

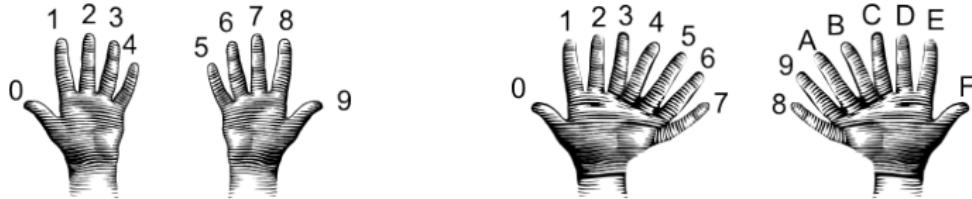
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
 - 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 i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):

- 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.

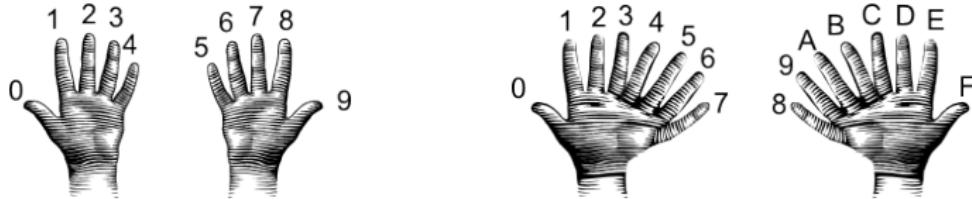
- Example: what is 99 as a decimal number?

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 (assuming two-digit numbers):

- 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?

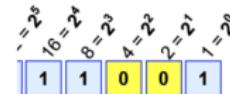
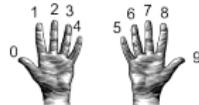
9 in decimal is 9. 9×16 is 144.

9 in decimal digits is 9

$144 + 9$ is 153.

Answer is 153.

Decimal to Binary: Converting Between Bases

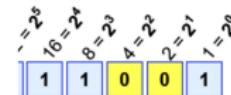
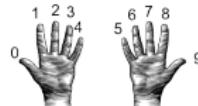


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From decimal to binary:

- Divide by $128 (= 2^7)$. Quotient is the first digit.

Decimal to Binary: Converting Between Bases

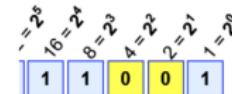
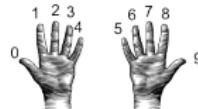


Example: $1 \times 16 + 1 \times 8 + 1 \times 4 + 1 \times 1 = 16 + 8 + 4 + 1 = 25$

- 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.

Decimal to Binary: Converting Between Bases

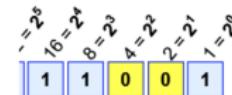
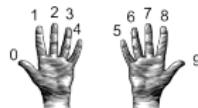


Example: $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 4 + 1 = 25$

- 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.

Decimal to Binary: Converting Between Bases

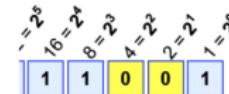
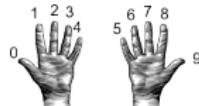


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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.

Decimal to Binary: Converting Between Bases

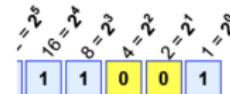


$$\text{Example: } 1 \times 16 + 1 \times 8 + 0 \times 4 + 1 \times 1 = 16 + 8 + 1 = 25$$

- 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.

Decimal to Binary: Converting Between Bases

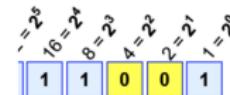
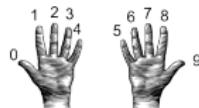


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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.

Decimal to Binary: Converting Between Bases

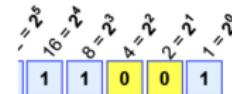
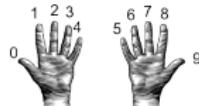


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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.

Decimal to Binary: Converting Between Bases

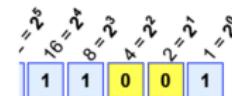
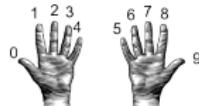


$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- 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

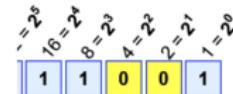
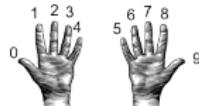


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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



$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

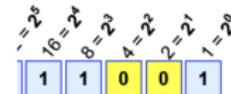
- From decimal to binary:

- 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



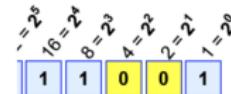
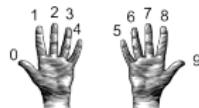
- From decimal to binary:

- 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



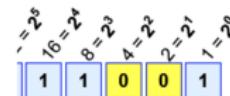
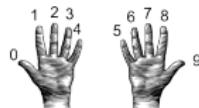
- From decimal to binary:

- 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?

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

2/64 is 0 rem 2.

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

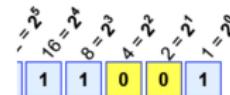
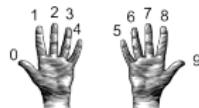
- From decimal to binary:

- 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:

Decimal to Binary: Converting Between Bases



$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1 = 16 + 8 + 4 + 2 + 1 = 25$$

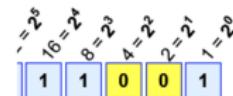
- From decimal to binary:

- 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



$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 4 + 1 \times 1 = 16 + 8 + 1 = 25$$

- From decimal to binary:

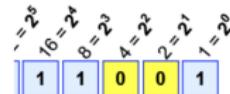
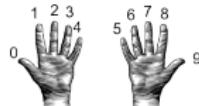
- Divide by $128 (= 2^7)$. Quotient is the first digit.
- Divide remainder by $64 (= 2^6)$. Quotient is the next digit.
- Divide remainder by $32 (= 2^5)$. Quotient is the next digit.
- Divide remainder by $16 (= 2^4)$. Quotient is the next digit.
- Divide remainder by $8 (= 2^3)$. Quotient is the next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



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From decimal to binary:

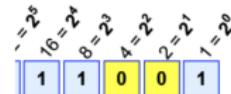
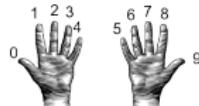
- Divide by $128 (= 2^7)$. Quotient is the first digit.
- Divide remainder by $64 (= 2^6)$. Quotient is the next digit.
- Divide remainder by $32 (= 2^5)$. Quotient is the next digit.
- Divide remainder by $16 (= 2^4)$. Quotient is the next digit.
- Divide remainder by $8 (= 2^3)$. Quotient is the next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



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From decimal to binary:

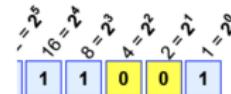
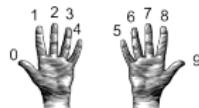
- Divide by $128 (= 2^7)$. Quotient is the first digit.
- Divide remainder by $64 (= 2^6)$. Quotient is the next digit.
- Divide remainder by $32 (= 2^5)$. Quotient is the next digit.
- Divide remainder by $16 (= 2^4)$. Quotient is the next digit.
- Divide remainder by $8 (= 2^3)$. Quotient is the next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

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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?

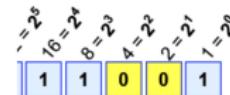
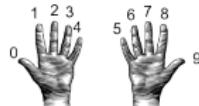
130/128 is 1 rem 2. First digit is 1: 1...

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.

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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?

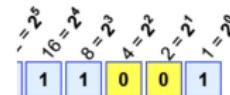
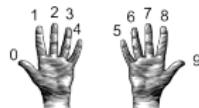
130/128 is 1 rem 2. First digit is 1: 1...

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



$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 4 + 1 \times 1 = 16 + 8 + 1 = 25$$

- 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?

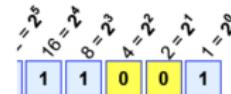
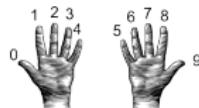
130/128 is 1 rem 2. First digit is 1: 1...

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



$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 4 + 1 \times 1 = 16 + 8 + 1 = 25$$

- 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...

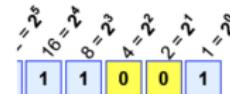
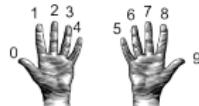
2/64 is 0 rem 2. Next digit is 0: 10...

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



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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...

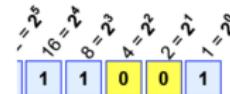
2/64 is 0 rem 2. Next digit is 0: 10...

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:

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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...

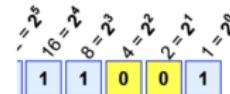
2/64 is 0 rem 2. Next digit is 0: 10...

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



$$\text{Example: } 1 \times 16 + 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 = 25$$

- 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...

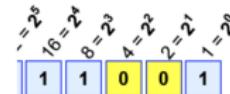
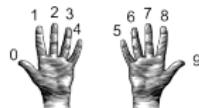
2/32 is 0 rem 2. Next digit is 0: 100...

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



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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...

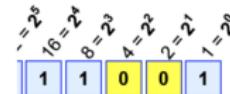
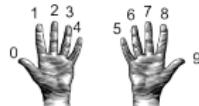
2/32 is 0 rem 2. Next digit is 0: 100...

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



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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...

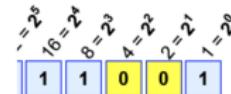
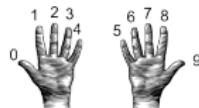
2/32 is 0 rem 2. Next digit is 0: 100...

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



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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...

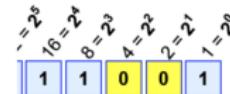
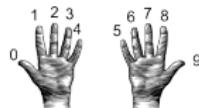
2/16 is 0 rem 2. Next digit is 0: 1000...

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.

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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...

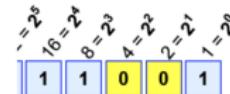
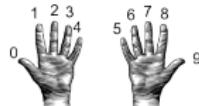
2/16 is 0 rem 2. Next digit is 0: 1000...

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:

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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...

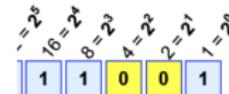
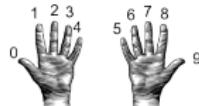
2/16 is 0 rem 2. Next digit is 0: 1000...

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...

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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...

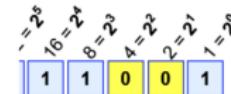
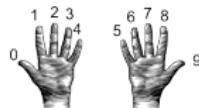
2/8 is 0 rem 2. Next digit is 0: 10000...

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

Decimal to Binary: Converting Between Bases



$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- 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...

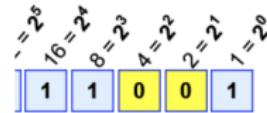
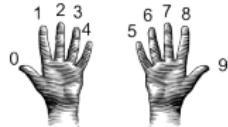
2/8 is 0 rem 2. Next digit is 0: 10000...

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

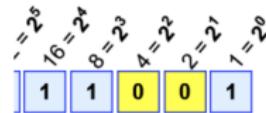
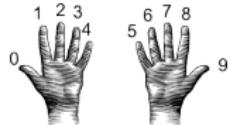
Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

Decimal to Binary: Converting Between Bases

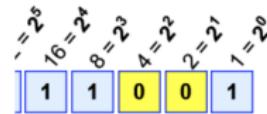


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

$99 / 128$ is 0 rem 99.

Decimal to Binary: Converting Between Bases

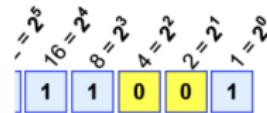


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0:

Decimal to Binary: Converting Between Bases



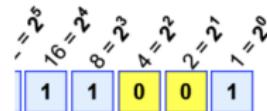
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35.

Decimal to Binary: Converting Between Bases



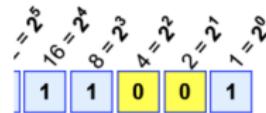
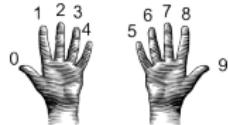
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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:

Decimal to Binary: Converting Between Bases



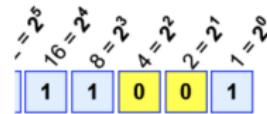
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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: $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 4 + 1 = 25$

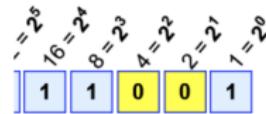
- Example: what is 99 in binary notation?

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.

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

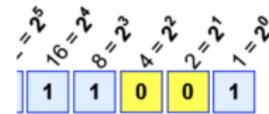
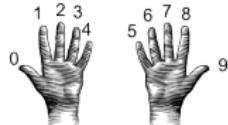
- Example: what is 99 in binary notation?

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:

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

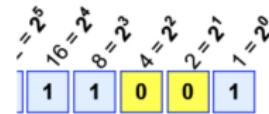
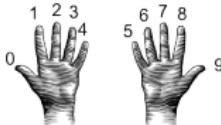
- Example: what is 99 in binary notation?

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...

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1 = 16 + 8 + 4 + 2 + 1 = 25$

- Example: what is 99 in binary notation?

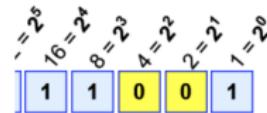
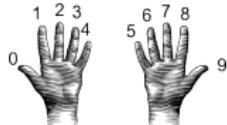
99/128 is 0 rem 99. First digit is 0: 0...

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: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

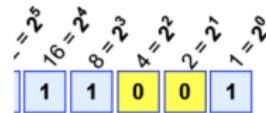
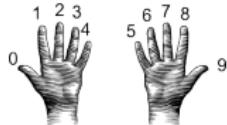
99/128 is 0 rem 99. First digit is 0: 0...

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:

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

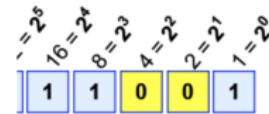
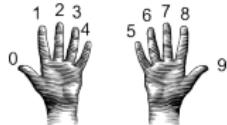
99/128 is 0 rem 99. First digit is 0: 0...

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...

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

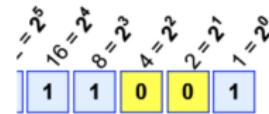
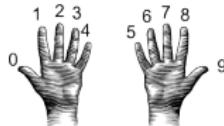
99/64 is 1 rem 35. Next digit is 1: 01...

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.

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

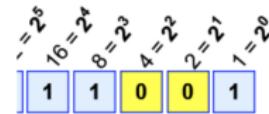
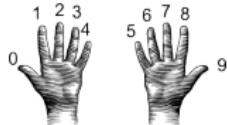
99/64 is 1 rem 35. Next digit is 1: 01...

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:

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

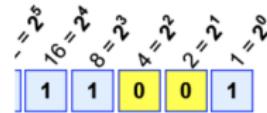
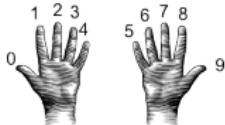
99/64 is 1 rem 35. Next digit is 1: 01...

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...

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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...

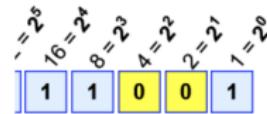
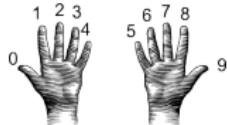
35/32 is 1 rem 3. Next digit is 1: 011...

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.

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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...

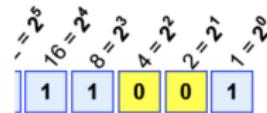
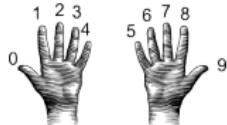
35/32 is 1 rem 3. Next digit is 1: 011...

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: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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...

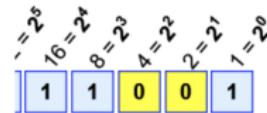
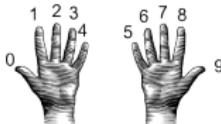
35/32 is 1 rem 3. Next digit is 1: 011...

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...

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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...

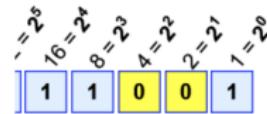
3/16 is 0 rem 3. Next digit is 0: 0110...

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.

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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...

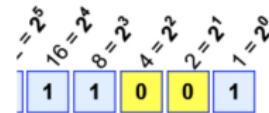
3/16 is 0 rem 3. Next digit is 0: 0110...

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:

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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...

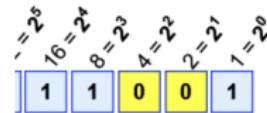
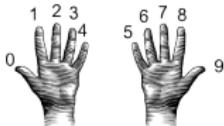
3/16 is 0 rem 3. Next digit is 0: 0110...

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...

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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...

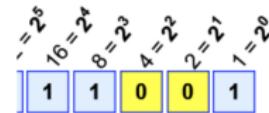
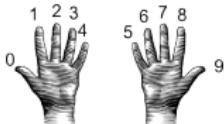
3/8 is 0 rem 3. Next digit is 0: 01100...

3/4 is 0 remainder 3. Next digit is 0: 011000...

3/2 is 1 rem 1. Next digit is 1: 0110001...

Adding the last remainder: 01100011

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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...

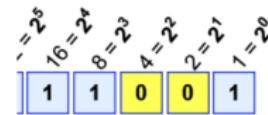
3/4 is 0 remainder 3. Next digit is 0: 011000...

3/2 is 1 rem 1. Next digit is 1: 0110001...

Adding the last remainder: 01100011

Answer is 1100011.

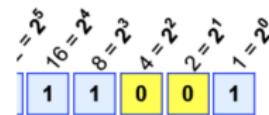
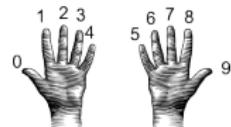
Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From binary to decimal:
 - Set sum = last digit.

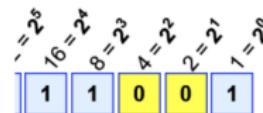
Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 4 + 0 + 1 = 25$

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

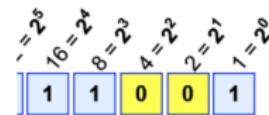
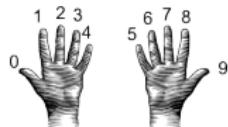
Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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.

Binary to Decimal: Converting Between Bases

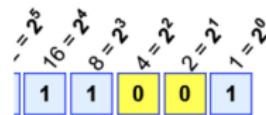
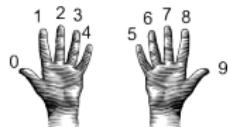


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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.

Binary to Decimal: Converting Between Bases

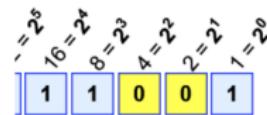
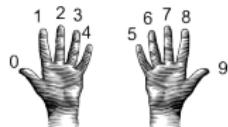


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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.

Binary to Decimal: Converting Between Bases

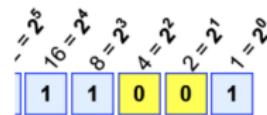
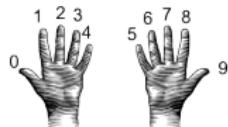


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by 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.

Binary to Decimal: Converting Between Bases

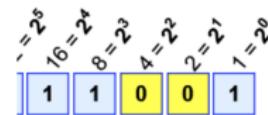
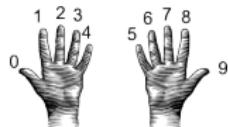


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by 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.

Binary to Decimal: Converting Between Bases

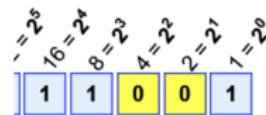
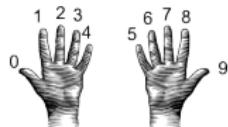


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by 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.

Binary to Decimal: Converting Between Bases

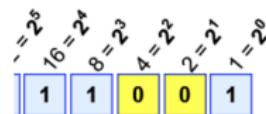


Example: $1 \times 16 + 1 \times 8 + 1 \times 4 + 1 \times 1 = 16 + 8 + 4 + 1 = 25$

- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by 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.

Binary to Decimal: Converting Between Bases



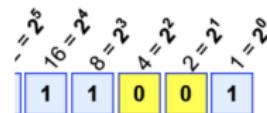
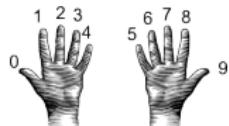
Example: $1 \times 16 + 1 \times 8 + 1 \times 4 + 1 \times 1 = 16 + 8 + 4 + 1 = 25$

- 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:

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 4 + 0 + 1 = 25$

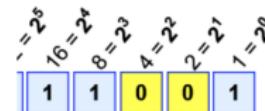
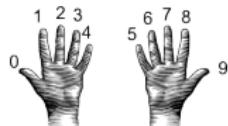
- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by 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

$0 \times 2 = 0$. Add 0 to sum:

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

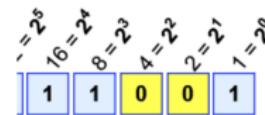
- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by 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



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From binary to decimal:

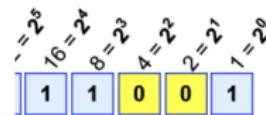
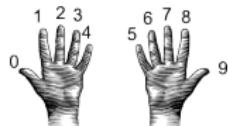
- Set sum = last digit.
- Multiply next digit by $2 = 2^1$. Add to sum.
- Multiply next digit by $4 = 2^2$. Add to sum.
- Multiply next digit by $8 = 2^3$. Add to sum.
- Multiply next digit by $16 = 2^4$. Add to sum.
- Multiply next digit by $32 = 2^5$. Add to sum.
- Multiply next digit by $64 = 2^6$.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

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 4 + 0 + 1 = 25$

- From binary to decimal:

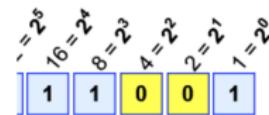
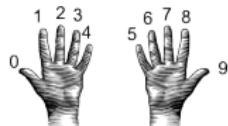
- Set sum = last digit.
- Multiply next digit by 2^1 . Add to sum.
- Multiply next digit by 2^2 . Add to sum.
- Multiply next digit by 2^3 . Add to sum.
- Multiply next digit by 2^4 . Add to sum.
- Multiply next digit by 2^5 . Add to sum.
- Multiply next digit by 2^6 . Add to sum.
- Multiply next digit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

Sum starts with: 1

$0 \times 2 = 0$. Add 0 to sum: 1

$1 \times 4 = 4$. Add 4 to sum: 5

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by 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?

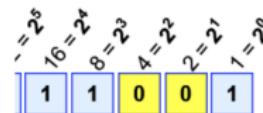
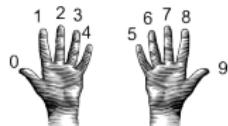
Sum starts with: 1

$0 \times 2 = 0$. Add 0 to sum: 1

$1 \times 4 = 4$. Add 4 to sum: 5

$1 \times 8 = 8$. Add 8 to sum:

Binary to Decimal: Converting Between Bases



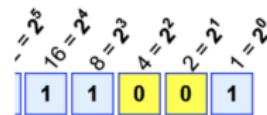
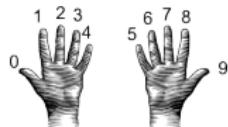
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by 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

Binary to Decimal: Converting Between Bases



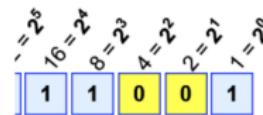
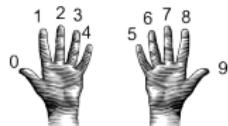
Example: $1 \times 16 + 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 1 = 25$

- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by 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
 $0 \times 2 = 0$. Add 0 to sum: 1
 $1 \times 4 = 4$. Add 4 to sum: 5
 $1 \times 8 = 8$. Add 8 to sum: 13
 $1 \times 16 = 16$. Add 16 to sum:

Binary to Decimal: Converting Between Bases



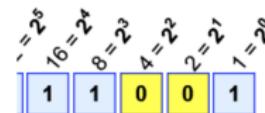
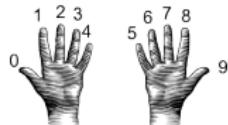
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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



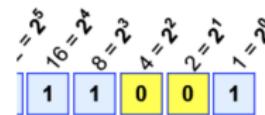
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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



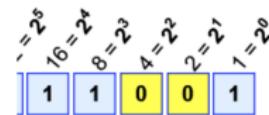
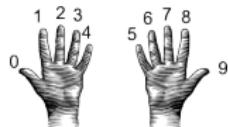
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- 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

Binary to Decimal: Converting Between Bases



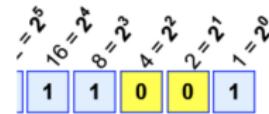
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by 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
$0 \times 2 = 0$. Add 0 to sum:	1
$1 \times 4 = 4$. 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

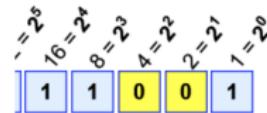
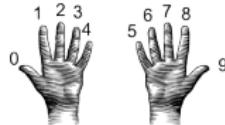


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with:

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

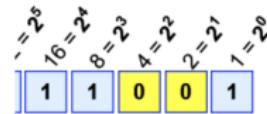
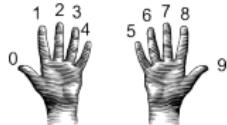
- Example: What is 10100100 in decimal?

Sum starts with:

0

$0 \times 2 = 0.$ Add 0 to sum:

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

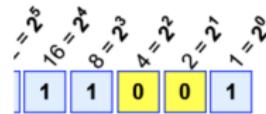
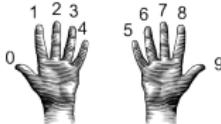
Sum starts with:

0

$0 \times 2 = 0.$ Add 0 to sum:

0

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

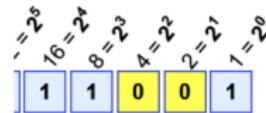
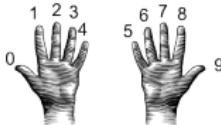
- Example: What is 10100100 in decimal?

Sum starts with: 0

$0 \times 2 = 0$. Add 0 to sum: 0

$1 \times 4 = 4$. Add 4 to sum:

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

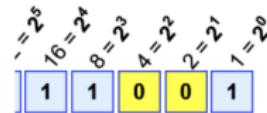
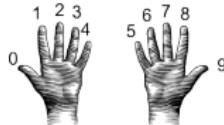
- Example: What is 10100100 in decimal?

Sum starts with: 0

$0 \times 2 = 0$. Add 0 to sum: 0

$1 \times 4 = 4$. Add 4 to sum: 4

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 4 + 0 + 1 = 25$

- Example: What is 10100100 in decimal?

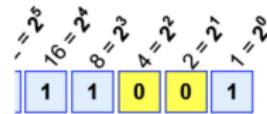
Sum starts with: 0

$0 \times 2 = 0$. Add 0 to sum: 0

$1 \times 4 = 4$. Add 4 to sum: 4

$0 \times 8 = 0$. Add 0 to sum:

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

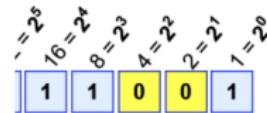
Sum starts with: 0

$0 \times 2 = 0$. Add 0 to sum: 0

$1 \times 4 = 4$. Add 4 to sum: 4

$0 \times 8 = 0$. Add 0 to sum: 4

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0

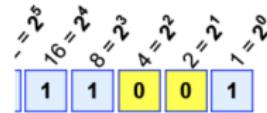
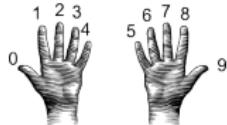
$0 \times 2 = 0$. Add 0 to sum: 0

$1 \times 4 = 4$. Add 4 to sum: 4

$0 \times 8 = 0$. Add 0 to sum: 4

$0 \times 16 = 0$. Add 0 to sum:

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0

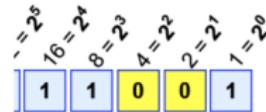
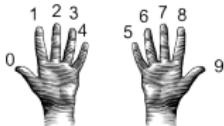
$0 \times 2 = 0$. Add 0 to sum: 0

$1 \times 4 = 4$. Add 4 to sum: 4

$0 \times 8 = 0$. Add 0 to sum: 4

$0 \times 16 = 0$. Add 0 to sum: 4

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0

$0 \times 2 = 0$. Add 0 to sum: 0

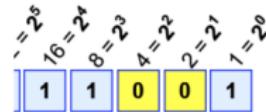
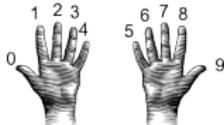
$1 \times 4 = 4$. Add 4 to sum: 4

$0 \times 8 = 0$. Add 0 to sum: 4

$0 \times 16 = 0$. Add 0 to sum: 4

$1 \times 32 = 32$. Add 32 to sum:

Binary to Decimal: Converting Between Bases

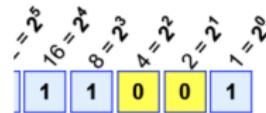
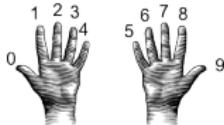


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

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

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0

$0 \times 2 = 0$. Add 0 to sum: 0

$1 \times 4 = 4$. Add 4 to sum: 4

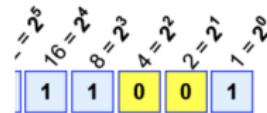
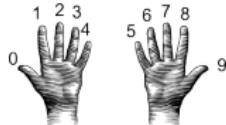
$0 \times 8 = 0$. Add 0 to sum: 4

$0 \times 16 = 0$. Add 0 to sum: 4

$1 \times 32 = 32$. Add 32 to sum: 36

$0 \times 64 = 0$. Add 0 to sum:

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0

$0 \times 2 = 0$. Add 0 to sum: 0

$1 \times 4 = 4$. Add 4 to sum: 4

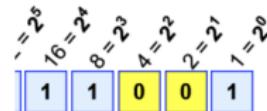
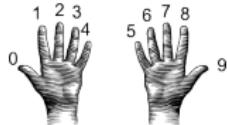
$0 \times 8 = 0$. Add 0 to sum: 4

$0 \times 16 = 0$. Add 0 to sum: 4

$1 \times 32 = 32$. Add 32 to sum: 36

$0 \times 64 = 0$. Add 0 to sum: 36

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 4 + 1 \times 1 = 16 + 8 + 4 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0

$0 \times 2 = 0$. Add 0 to sum: 0

$1 \times 4 = 4$. Add 4 to sum: 4

$0 \times 8 = 0$. Add 0 to sum: 4

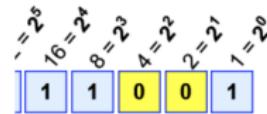
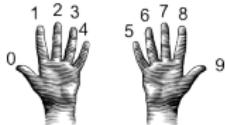
$0 \times 16 = 0$. Add 0 to sum: 4

$1 \times 32 = 32$. Add 32 to sum: 36

$0 \times 64 = 0$. Add 0 to sum: 36

$1 \times 128 = 0$. Add 128 to sum:

Binary to Decimal: Converting Between Bases

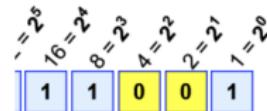
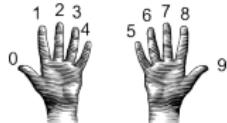


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

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

Binary to Decimal: Converting Between Bases



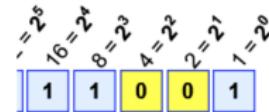
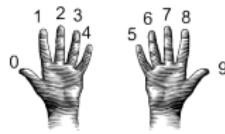
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

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

The answer is 164.

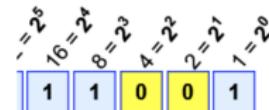
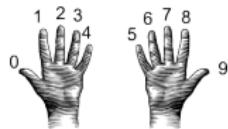
Design Challenge: Incrementers



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

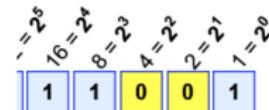
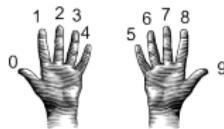
Design Challenge: Incrementers



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

Design Challenge: Incrementers

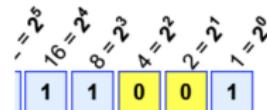
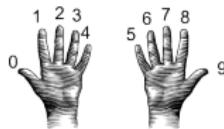


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

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

Design Challenge: Incrementers



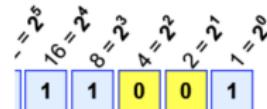
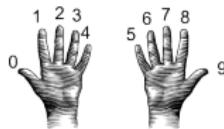
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

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

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

Design Challenge: Incrementers



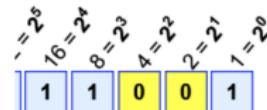
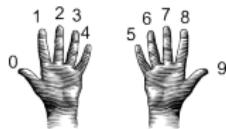
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

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

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

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

Design Challenge: Incrementers



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

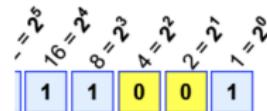
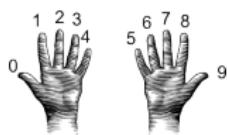
- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

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

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

Hint: Convert to numbers, increment, and convert back to strings.

Design Challenge: Incrementers



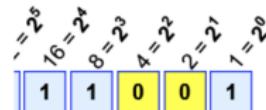
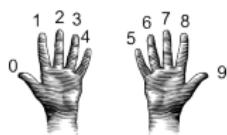
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

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

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

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

Design Challenge: Incrementers



$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

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

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

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

Hint: Convert to numbers, increment, and convert back to strings.

- Challenge: Write an algorithm for incrementing binary numbers.

Example: "1001" → "1010"

Recap



- 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.

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 during the time of the exam

Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only during the time of the exam
- 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 during the time of the exam
- 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 during the time of the exam
- 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 during the time of the exam
- 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.

Final Overview: Format

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

Final Overview: Format

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

Final Overview: Format

- Although the exam is remote, we still suggest you prepare 1 piece of **8.5" x 11"** paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ Best if you design/write yours since excellent way to study.

Final Overview: Format

- Although the exam is remote, we still suggest you prepare 1 piece of **8.5" x 11"** paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ Best if you design/write yours since excellent way to study.
 - ▶ Avoid scrambling through web searches and waste time during 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.
 - ▶ Avoid scrambling through web searches and waste time during the exam.
- 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.
 - ▶ Avoid scrambling through web searches and waste time during the exam.
- The exam format:
 - ▶ Like a long Lab Quiz, you scroll down to answer all questions.

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.
 - ▶ Avoid scrambling through web searches and waste time during the exam.
- The exam format:
 - ▶ Like a long Lab Quiz, you scroll down to answer all questions.
 - ▶ Questions roughly correspond to the 10 parts from old exams, but will appear as a larger number of questions on Gradescope
 - ▶ Questions 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.
 - ▶ Avoid scrambling through web searches and waste time during the exam.
- The exam format:
 - ▶ Like a long Lab Quiz, you scroll down to answer all questions.
 - ▶ Questions roughly correspond to the 10 parts from old exams, but will appear as a larger number of questions on Gradescope
 - ▶ Questions are variations on the programming assignments, lab exercises, and lecture design challenges.
- 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

19 December 2008

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 that contains notes, programs, or formulas.
- You may take breaks; you may have water, pens and pencils, and your note sheet.
- You are not allowed to 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 or communism, obtaining other advantage, and fabrication of records and official documents) as serious offenses against the integrity of the educational process. The student handbook of CUNY defines academic honesty and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Policy.

I acknowledge that all cases of academic dishonesty will be reported to the Office of Student and Academic Conduct.	
Name:	
Signature:	
Email:	
Signature:	

Exam Options

Exam Times:

- Default: Regular Time: Monday, 14 December, 9-10:30am.

FINAL EXAM, VERSION 3
CSci 127: Introduction to Computer Science
Hunter College, City University of New York

19 December 2018

Exam Rules

- More of 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 that contains prewritten programs.
- You may take notes; you may have up to one pen and pencil, 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 or communism, obtaining other advantage, and fabrication of records and official documents) as serious offenses against the integrity of the educational process. The student handbook at CUNY.org provides more information about what constitutes academic dishonesty according to the Hunter College Student Handbook.

I acknowledge that all forms of academic dishonesty will be reported to the Office of Student and Academic Conduct.	
Name:	
Signature:	
Email:	
Signature:	

Exam Options

Exam Times:

- Default: Regular Time: Monday, 14 December, 9-10:30am.
- Alternate Time: Reading Day, Friday, 11 December, 8am-10am.

FINAL EXAM, VERSION 3
CSci 127: Introduction to Computer Science
Hunter College, City University of New York

19 December 2018

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 1/2" x 11" piece of paper that contains prewritten formulas.
- You may take breaks; you may leave the room for a pen and pencil, and your water bottle.
- 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 or communism, obtaining other advantage, and fabrication of records and official documents) as serious offenses that violate the principles of the college. The student handbook of CUNY defines acts of academic honesty and will pursue cases of academic dishonesty according to the Hunter College Student Handbook.

I acknowledge that all cases of academic dishonesty will be reported to the Office of Student and Academic Conduct.	
Name:	
Signature:	
Email:	
Signature:	

Exam Options

Exam Times:

- Default: Regular Time: Monday, 14 December, 9-10:30am.
- Alternate Time: Reading Day, Friday, 11 December, 8am-10am.
- Accessibility Testing: If you have not done so already, email me no later than 7 December.

FINAL EXAM, VERSION 3
CSci 127: Introduction to Computer Science
Hunter College, City University of New York

19 December 2020

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 that contains programs and formulas.
- You may take breaks; you must have at least one pen and pencil, 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 or communism, obtaining other advantage, and fabrication of records and official documents) as serious offenses against the integrity of the educational process. The student handbook of the City University of New York defines the rules of academic honesty and will pursue cases of academic dishonesty according to the Hunter College Student Handbook.

I acknowledge that all cases of academic dishonesty will be reported to the Office of Student and Academic Conduct.	
Name:	
Signature:	
Email:	
Signature:	

Exam Options

Exam Times:

- Default: Regular Time: Monday, 14 December, 9-10:30am.
- Alternate Time: Reading Day, Friday, 11 December, 8am-10am.
- Accessibility Testing: If you have not done so already, email me no later than 7 December.
- Survey for your choice will be available next lecture. **No survey answer implies you will take the exam on 14 December.**

FINAL EXAM, VERSION 3
CSci 127: Introduction to Computer Science
Hunter College, City University of New York

19 December 2020

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 that contains programs and formulas.
- You may bring a calculator, your own laptop or two pens and pencils, and your note sheet.
- You may not use a computer, telephone, radio, sound system, 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 or communism, obtaining other advantage, and fabrication of records and official documents) as serious offenses against the integrity of the educational process. The Hunter College Code of Conduct governs academic honesty and will pursue cases of academic dishonesty according to the Hunter College Academic Honesty Policy.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating or communism, obtaining other advantage, and fabrication of records and official documents) as serious offenses against the integrity of the educational process. The Hunter College Code of Conduct governs academic honesty and will pursue cases of academic dishonesty according to the Hunter College Academic Honesty Policy.	
Name:	
Social Security:	
Email:	
Signature:	

Exam Options

Exam Times:

- Default: Regular Time: Monday, 14 December, 9-10:30am.
- Alternate Time: Reading Day, Friday, 11 December, 8am-10am.
- Accessibility Testing: If you have not done so already, email me no later than 7 December.
- Survey for your choice will be available next lecture. **No survey answer implies you will take the exam on 14 December.**

FINAL EXAM, VERSION 3
CSci 127: Introduction to Computer Science
Hunter College, City University of New York

19 December 2020

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 that contains programs and formulas.
- You may bring a calculator, your own laptop or two pens and pencils, and your note sheet.
- You may not use a computer, telephone, radio, sound system, 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 or communism, obtaining other advantage, and fabrication of records and official documents) as serious offenses against the integrity of the educational process. The Hunter College Code of Conduct regarding academic honesty and policies on cases of academic dishonesty according to the Hunter College website.

I acknowledge that all cases of academic dishonesty will be reported to the Office of Student and Academic Conduct.	
Name:	
Social Security:	
Email:	
Signature:	

Grading Options:

Exam Options

Exam Times:

- Default: Regular Time: Monday, 14 December, 9-10:30am.
- Alternate Time: Reading Day, Friday, 11 December, 8am-10am.
- Accessibility Testing: If you have not done so already, email me no later than 7 December.
- Survey for your choice will be available next lecture. **No survey answer implies you will take the exam on 14 December.**

FINAL EXAM, VERSION 3
CSci 122: Introduction to Computer Science
Hunter College, City University of New York

19 December 2020

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 that contains formulas, programs, or other relevant information. The paper must be CLEARED for academic honesty and will serve as proof of academic honesty according to the Hunter College Academic Honesty Policy.
- You may bring a calculator, reference tables, answer sheet, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (i.e., plagiarism, cheating or communism, obtaining other advantage, and fabrication of records and official documents) as serious offenses against the integrity of the educational process. The student handbook provides details on academic honesty and will serve as proof of academic dishonesty according to the Hunter College Academic Honesty Policy.

I acknowledge that all work on this exam is my own work and is equivalent to the work of Hunter and will result in zero points.	
Name:	
Social:	
Email:	
Signature:	

Grading Options:

- Default: Letter Grade.
- Credit/NoCredit grade— check with academic advisor and fill out form by **25 November**

Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab

Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Optional - attend [live Lab Review on Wednesday 11-12:30pm](#)

Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Optional - attend [live Lab Review on Wednesday 11-12:30pm](#)
- Take the Lab Quiz on Gradescope by 6pm on Wednesday

Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Optional - attend [live Lab Review on Wednesday 11-12:30pm](#)
- Take the Lab Quiz on Gradescope by 6pm on Wednesday
- Submit this week's 3 programming assignments (programs 50-52)

Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Optional - attend [live Lab Review on Wednesday 11-12:30pm](#)
- Take the Lab Quiz on Gradescope by 6pm on Wednesday
- Submit this week's 3 programming assignments (programs 50-52)
- At any point, visit our [Drop-In Tutoring 11am-5pm](#) for help!!!

Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Optional - attend [live Lab Review on Wednesday 11-12:30pm](#)
- Take the Lab Quiz on Gradescope by 6pm on Wednesday
- Submit this week's 3 programming assignments (programs 50-52)
- At any point, visit our [Drop-In Tutoring 11am-5pm](#) for help!!!
- Take the Lecture Preview on Blackboard on Monday (or no later than 10am on Tuesday)