

CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

- This lecture will be recorded

Announcements

- Great mentoring opportunity available!
Please check out BB announcement from
yesterday 4/19



Announcements

- Great mentoring opportunity available! Please check out BB announcement from yesterday 4/19
- I will be recruiting UTAs at the end of this term. You will hear from me by email after the final exam if you earn an A (+), **if you are interested keep an eye out for my emails.**



Announcements



- Great mentoring opportunity available! Please check out BB announcement from yesterday 4/19
- I will be recruiting UTAs at the end of this term. You will hear from me by email after the final exam if you earn an A (+), **if you are interested keep an eye out for my emails.**
- What should you do?

Announcements



- Great mentoring opportunity available! Please check out BB announcement from yesterday 4/19
- I will be recruiting UTAs at the end of this term. You will hear from me by email after the final exam if you earn an A (+), **if you are interested keep an eye out for my emails.**
- What should you do?
 - ▶ Get an A (+)

Announcements



- Great mentoring opportunity available! Please check out BB announcement from yesterday 4/19
- I will be recruiting UTAs at the end of this term. You will hear from me by email after the final exam if you earn an A (+), **if you are interested keep an eye out for my emails.**
- What should you do?
 - ▶ Get an A (+)
 - ▶ Be very comfortable with our labs and programming assignments

Announcements



- Great mentoring opportunity available! Please check out BB announcement from yesterday 4/19
- I will be recruiting UTAs at the end of this term. You will hear from me by email after the final exam if you earn an A (+), **if you are interested keep an eye out for my emails.**
- What should you do?
 - ▶ Get an A (+)
 - ▶ Be very comfortable with our labs and programming assignments
 - ▶ Be able to describe a successful tutoring experience (go to tutoring!!!)

Announcements



- Great mentoring opportunity available! Please check out BB announcement from yesterday 4/19
- I will be recruiting UTAs at the end of this term. You will hear from me by email after the final exam if you earn an A (+), **if you are interested keep an eye out for my emails.**
- What should you do?
 - ▶ Get an A (+)
 - ▶ Be very comfortable with our labs and programming assignments
 - ▶ Be able to describe a successful tutoring experience (go to tutoring!!!)
 - ▶ Previous tutoring experience helpful but not expected

Frequently Asked Questions

From email and tutoring.

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

Frequently Asked Questions

From email and tutoring.

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

The official final is Monday, 24 May, 9-11am.

Frequently Asked Questions

From email and tutoring.

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

The official final is Monday, 24 May, 9-11am.

The early final exam (alternative date) is on Friday, 21 May, 8-10am.

Frequently Asked Questions

From email and tutoring.

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

The official final is Monday, 24 May, 9-11am.

The early final exam (alternative date) is on Friday, 21 May, 8-10am.

Instead of a review sheet, we have:

Frequently Asked Questions

From email and tutoring.

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

The official final is Monday, 24 May, 9-11am.

The early final exam (alternative date) is on Friday, 21 May, 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.

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

The official final is Monday, 24 May, 9-11am.

The early final exam (alternative date) is on Friday, 21 May, 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.

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

The official final is Monday, 24 May, 9-11am.

The early final exam (alternative date) is on Friday, 21 May, 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 during our last meeting on 11 May.*

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 Python code editor interface. The left pane displays the code file `main.py` with the following content:

```
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)
```

The right pane has two tabs: `Result` and `Instructions`. The `Result` tab shows the output of the program, which is a regular hexagon drawn in purple. 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.

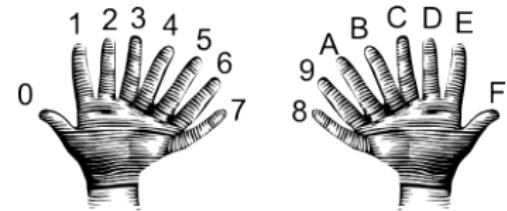
Week 2: variables, data types, more on loops & range()

- 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)
```

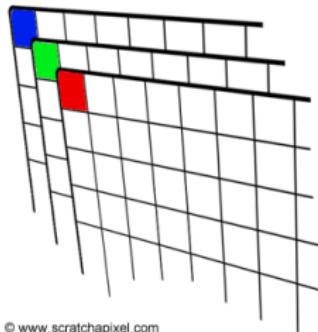
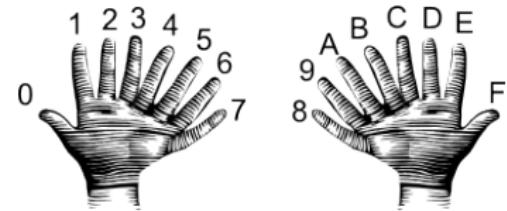
Week 3: colors, hex, slices, numpy & images

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



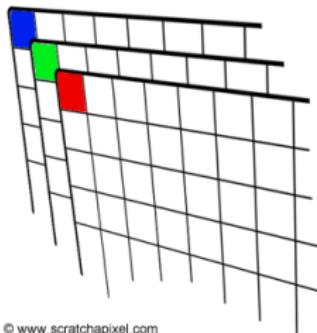
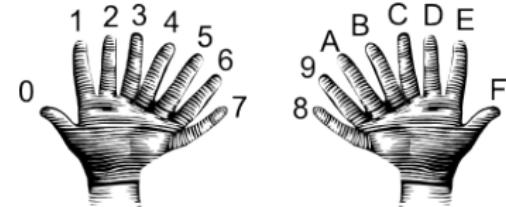
Week 3: colors, hex, slices, numpy & images

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.")

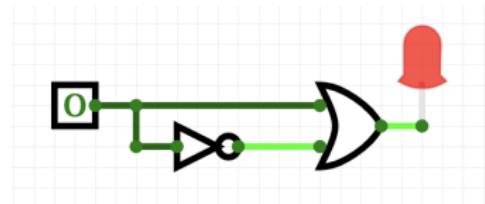
visibility = 0.2
winds = 40
conditions = "blowing snow"
if (winds > 35) and (visibility < 0.25) and \
    (conditions == "blowing snow" or conditions == "heavy snow"):
    print("Blizzard!")
```

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
1771,21843,36234,,2847,28423
1790,33131,4549,6159,1781,3827,49447
1800,60515,5740,6442,1755,4543,75955
1810,67541,6250,6840,2032,49734
1820,123704,11487,8246,2792,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312110,120113,140348,5346,10965,391114
1850,355441,128540,189540,5815,11534,41115
1860,613469,279122,32903,23593,25492,174777
1870,942292,419921,45468,37393,33029,1479103
1880,1164473,599403,5653,51980,33091,1911801
1890,1367111,700000,65000,56000,33091,2140134
1900,1850593,116582,152999,200567,67621,2437202
1910,2233142,1634351,284041,430980,8569,4766803
1920,2211103,2018354,446000,720201,11650,50048
1930,2067123,1979128,352543,45872,11582,4930446
1940,1889924,2498285,1297634,1394711,174441,7454995
1950,1960101,2738175,1550949,1451277,191555,7891957
1960,1690101,2738175,1899049,1451277,191555,7891984
1970,1539231,2460701,1471071,135443,7071646
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
2015,1444518,2436733,2339150,1459444,474558,8059405

nycHistPop.csv

In Lab 6

Week 6: structured data, pandas, & more design

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

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
1771,21843,36231,,2847,28423
1790,33131,4549,6159,1781,3827,49447
1800,60515,5740,6442,1755,4543,75955
1810,67541,6254,6841,1755,4543,75934
1820,123704,11187,8246,2792,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,313110,18013,14081,5348,10965,391114
1850,355491,21890,18591,5348,10965,391115
1860,513469,279122,32903,23593,25492,174777
1870,942292,419921,45468,37393,33029,1479103
1880,1164473,59943,5653,51980,33029,1911801
1890,1367711,70000,6534,51980,33029,1911804
1900,185093,116582,152999,200567,67621,2437202
1910,2233142,1634351,284041,430980,8569,4766803
1920,2211103,2018354,446071,446071,73201,11651,50048
1930,1867137,1796128,1796128,15831,4930446
1940,1889924,2469285,1297634,1394711,174441,7454995
1950,1960101,2738175,1550949,1451277,191555,7891957
1960,1690131,2319319,1809011,1809011,1809011,781984
1970,1539231,2460712,1472701,1472701,135443,781984
1980,1426285,2230936,1891325,1168972,352121,7071639
1990,1487536,2300664,1951598,1203789,737977,7322564
2000,1537195,2485326,2229379,1332450,419728,8080879
2010,1583873,2504705,2272722,1385108,6417513,8175133
2015,1444018,2646733,2339150,1459444,474558,8056405

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  
1810,69111,6340,...,6442,1755,4543,75934  
1820,123704,11187,...,8246,2792,6135,152056  
1830,20589,20535,...,9049,3023,7082,242278  
1840,31150,10113,...,14081,5346,10965,391114  
1850,35549,...,12891,5346,10965,391114  
1860,...,613469,...,279122,23903,23993,25492,174777  
1870,...,942292,...,419921,45468,...,37393,...,33029,...,1479103  
1880,...,1164473,...,59943,...,5653,...,51980,...,39301,...,1911801  
1890,...,1364473,...,61843,...,5653,...,51980,...,39301,...,1911804  
1900,...,185093,...,116582,...,152999,...,200567,...,67921,...,2437202  
1910,...,233142,...,1634351,...,2841,...,430980,...,8569,...,476683  
1920,...,2231103,...,2018354,...,44601,...,72021,...,11651,...,593083  
1930,...,6671103,...,1796128,...,254543,...,15821,...,4930446  
1940,...,1889924,...,2469285,...,1297634,...,1394711,...,174441,...,7454995  
1950,...,1960101,...,2738175,...,1550949,...,1451277,...,191555,...,7891957  
1960,...,1660101,...,2738175,...,1550949,...,1451277,...,191555,...,7891957  
1970,...,1639231,...,2467011,...,1472701,...,135443,...,7071646  
1980,...,1426285,...,2230936,...,1891325,...,1168972,...,252121,...,7071639  
1990,...,1487536,...,2300664,...,1951598,...,1302789,...,737977,...,7322564  
2000,...,1537195,...,2485326,...,2229379,...,1332450,...,419728,...,8080879  
2010,...,1583873,...,2504705,...,2272722,...,1385108,...,474558,...,8175133  
2015,...,1444518,...,2546733,...,2339150,...,1459446,...,474558,...,8156405
```

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.....  
Five 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  
1810,70000,6350,7000,1800,5300,93734  
1820,123704,11187,8246,2792,6135,152056  
1830,20589,20535,9049,3023,7082,242278  
1840,31510,11013,14000,5348,10965,391114  
1850,35549,12800,18500,5850,13000,45115  
1860,813469,279122,23903,23933,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59943,5653,51980,33091,1911801  
1890,1380000,720000,68000,51800,38000,210134  
1900,1850093,116582,152999,200567,67621,2437202  
1910,2233142,1634351,2841,430980,8569,476683  
1920,22461103,2018354,44600,720201,116500,591088  
1930,26671128,2079128,40000,707128,158200,6930446  
1940,1889924,2690285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550949,1451277,191555,7991957  
1960,1690000,2319359,1809000,1400000,1300000,781984  
1970,1539231,2465070,187473,1472701,135443,798462  
1980,1426285,2230936,1891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1203789,378977,7322564  
2000,1537195,2485326,2229379,1332650,419782,8080879  
2010,1583873,2504705,2277722,1385108,4175133,8175133  
2015,1444018,2646733,2339150,1459446,474558,8059405
```

nycHistPop.csv

In Lab 6

Week 6: structured data, pandas, & more design

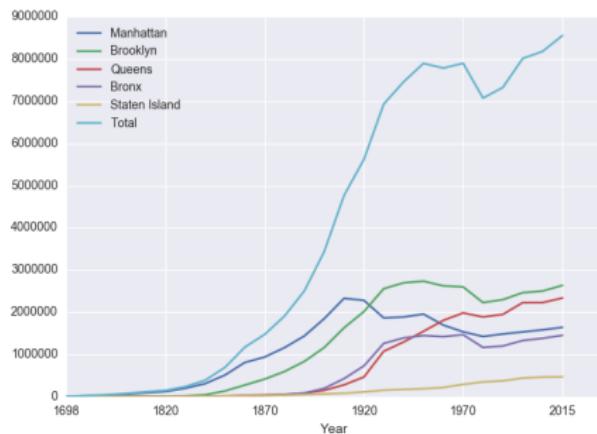
```
import matplotlib.pyplot as plt  
import pandas as pd
```

```
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  
1698,Manhattan,2037,727,7188  
1771,21843,36231,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,75955  
1810,70531,6211,6881,1811,4971,89734  
1820,123704,11187,8246,2792,6135,152056  
1830,202589,20535,9049,3023,7082,242278  
1840,312110,18013,14031,5348,10965,391114  
1850,355441,21800,18500,5835,12000,45115  
1860,813469,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59943,5653,51980,33051,1911801  
1890,1385111,72000,6500,57000,35000,2151134  
1900,1850993,1165852,152999,200567,67921,2437202  
1910,233142,1634351,2841,430980,8569,476683  
1920,2210103,2018354,44601,44601,73201,11651,50048  
1930,2667103,2407128,35254,35254,5830,5830446  
1940,1889924,2690285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550849,1451277,191555,7991957  
1960,1696101,2738175,1550849,1451277,191555,7991984  
1970,1539231,2465701,1471701,1471701,135443,7984650  
1980,1426285,2230936,11891325,1169792,352121,7071639  
1990,1487536,2300664,1951598,1320789,378977,7322564  
2000,1537195,2485326,2229379,1332450,419728,8080879  
2010,1583873,2504705,2216722,1385108,419728,8175133  
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

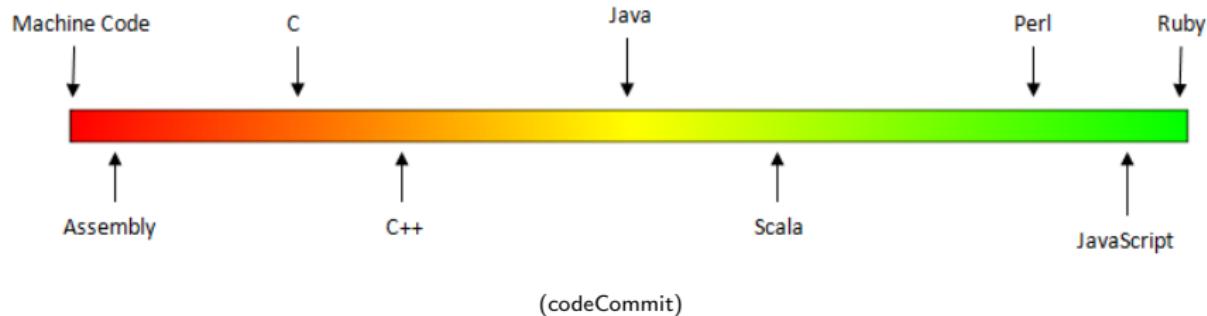
- Log-in to Gradescope
- Find LECTURE 11 Quiz
- Take the quiz
- **You have 3 minutes**

Today's Topics



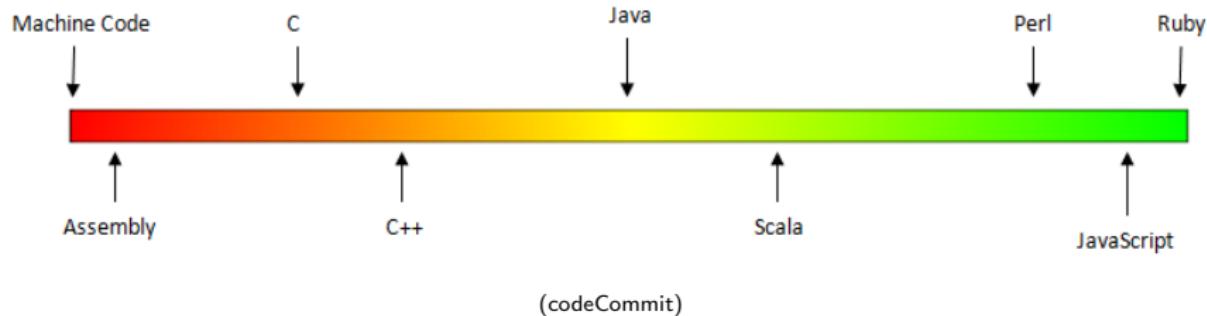
- Design Patterns: Searching
- Python Recap
- **Machine Language**
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- Final Exam: Format

Low-Level vs. High-Level Languages



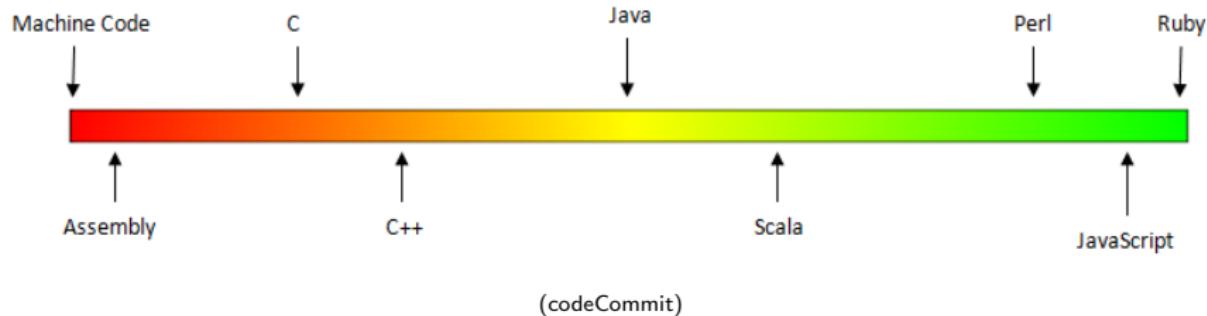
- Can view programming languages on a continuum.

Low-Level vs. High-Level Languages



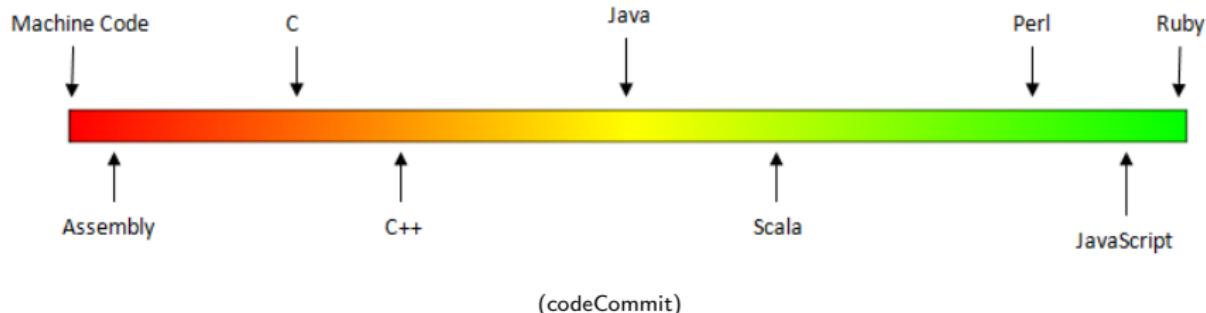
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages**

Low-Level vs. High-Level Languages



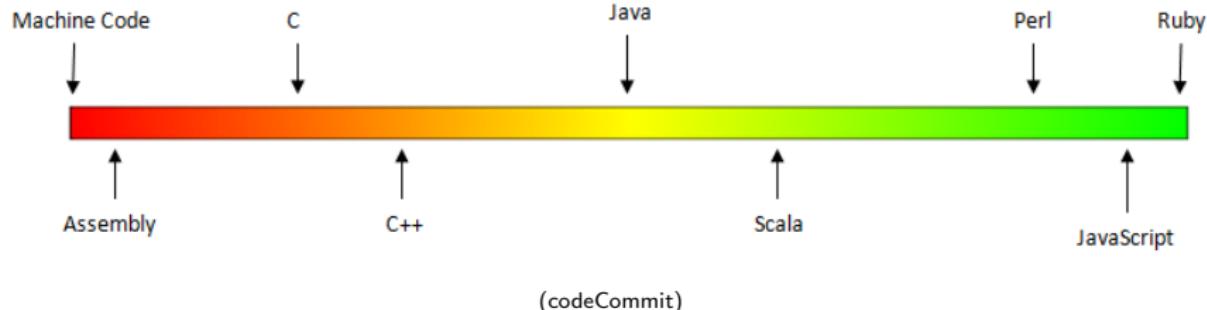
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).

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

Blindtext: 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 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.

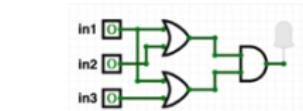


Data
&
Instructions

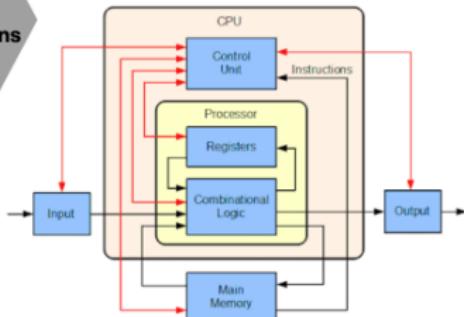
```
01110100011100100110000101  
1110010001101101110101011  
00111110001101000 101011  
00101110 00010000111111  
1110110111101000111011011  
001011010 101101 10011000  
01100101011010001000001  
11100100001101 00111101  
011000101101100011010101  
01000100000011000100 000  
0110010101 100110100101  
10010 1000000110110011101  
01101100110010000000000  
1110001000000000000000000  
011000110 10111010111001  
00011100 0000001100000000  
100 01000 01000 1000 101  
00100000011101000101011100  
00010 110010011 100100100  
11101000110111011011 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)
```

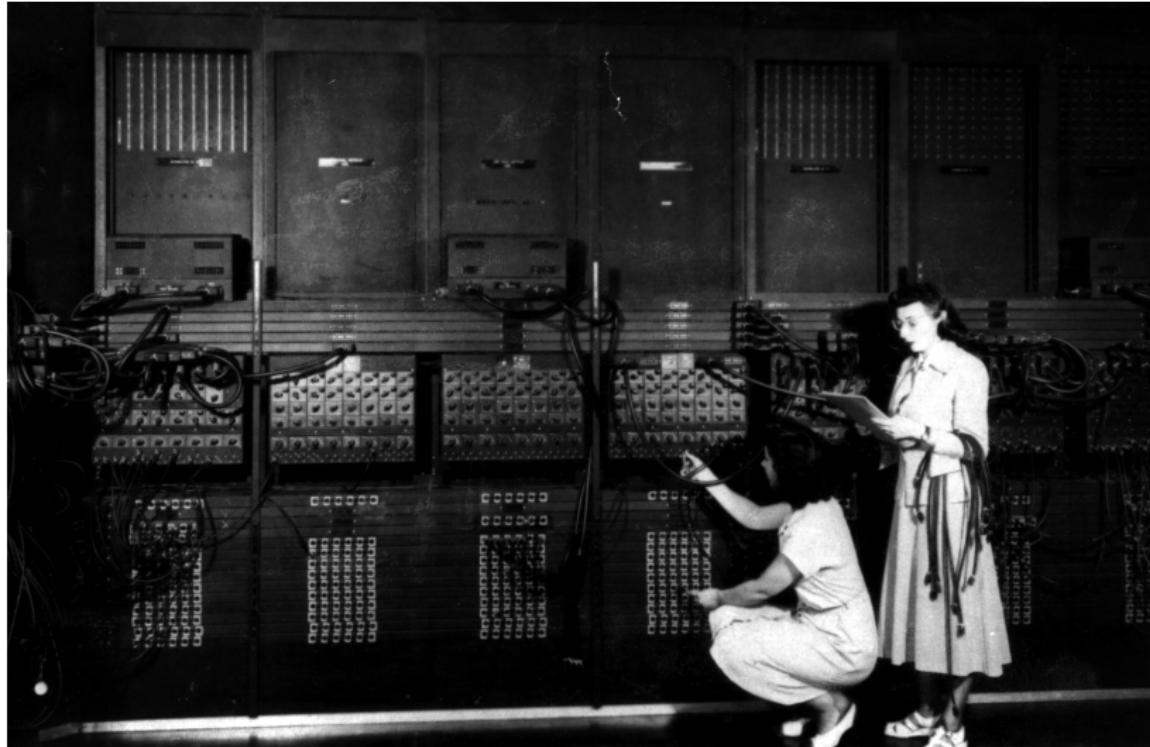
Data
&
Instructions



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.

A screenshot of a terminal window displaying assembly code and memory dump information. The assembly code is:

```
R 002000 C2 3B      REP    #438  
R 002002 1B          CLC  
R 002003 FB          SEI  
R 002004 69 34 12    LDH   #1234  
R 002007 69 21 43    ADC   #4324  
R 002008 8F 03 7F 01  STA   #017F03  
R 002009 00           CLD  
R 00200F E2 30      SEP   #38  
R 002011 89          SBC  
R 2012              BRK
```

The memory dump shows the state of registers and memory at address 2000:

PC	PC	Mem[2000..200B]	X	Y	SP	DP	RR
00 0012	00110000 0000 0000 0002 CFFF 0000 00	00000000	0000	0000	0000	0000	00

Below the dump is a BREAK instruction:

```
BREAK
```

Registers and memory dump details:

Reg	Value	Mem[2000..200B]
PC	0012	00110000 0000 0000 0002 CFFF 0000 00
Y	7FFF	00000000
RR	0000	00000000 5555 0000 0000 CFFF 0000 00
SP	0000	00000000 7FFF 0000 0000 0000 0000 0000 00
DP	0000	00000000 7FFF 0000 0000 0000 0000 0000 00
X	0000	00000000 7FFF 0000 0000 0000 0000 0000 00
Mem[2000..200B]	00000000 5555 0000 0000 CFFF 0000 0000 00	

(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 assembly code and its corresponding binary representation. The assembly code includes instructions like LDI, GCD, SED, ADC, STA, CLD, SRL, and MUL. The binary output consists of two columns of hex values.

PC	PC	Memory[PC]	A	X	Y	SP	DP	R
00	E012	00100000 0000 0000 0002 CFFF 0000 00						
00	2000							
BREAK								
00	E013	00100000 5555 0000 000C 1FFF 0000 00						
00	2003	7FFF						
00	00FF03	55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00						

(wiki)

Machine Language



A screenshot of a computer screen displaying assembly code and processor registers. The assembly code is shown in a text editor window, with each line containing an instruction address, the instruction itself, and its operands. The registers window shows various寄存器 (Registers) with their current values.

Register	Value
PC	00000000 00000000 00000000 00000000
R0	00000000 00000000 00000000 00000000
R1	00000000 00000000 00000000 00000000
R2	00000000 00000000 00000000 00000000
R3	00000000 00000000 00000000 00000000
R4	00000000 00000000 00000000 00000000
R5	00000000 00000000 00000000 00000000
R6	00000000 00000000 00000000 00000000
R7	00000000 00000000 00000000 00000000
R8	00000000 00000000 00000000 00000000
R9	00000000 00000000 00000000 00000000
R10	00000000 00000000 00000000 00000000
R11	00000000 00000000 00000000 00000000
R12	00000000 00000000 00000000 00000000
R13	00000000 00000000 00000000 00000000
R14	00000000 00000000 00000000 00000000
R15	00000000 00000000 00000000 00000000
R16	00000000 00000000 00000000 00000000
R17	00000000 00000000 00000000 00000000
R18	00000000 00000000 00000000 00000000
R19	00000000 00000000 00000000 00000000
R20	00000000 00000000 00000000 00000000
R21	00000000 00000000 00000000 00000000
R22	00000000 00000000 00000000 00000000
R23	00000000 00000000 00000000 00000000
R24	00000000 00000000 00000000 00000000
R25	00000000 00000000 00000000 00000000
R26	00000000 00000000 00000000 00000000
R27	00000000 00000000 00000000 00000000
R28	00000000 00000000 00000000 00000000
R29	00000000 00000000 00000000 00000000
R30	00000000 00000000 00000000 00000000
R31	00000000 00000000 00000000 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.

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 C0 30
ADD R0, R1, R2	00000002 1B 00 00
LDI R3, 1000	00000003 FB 00 00
ADD R3, R0, R1	00000004 34 12 00 00
LDI R4, 1000	00000005 34 12 00 00
ADD R4, R0, R1	00000006 21 43 00 00
LDI R5, 1000	00000007 34 12 00 00
ADD R5, R0, R1	00000008 21 43 00 00
LDI R6, 1000	00000009 BF 03 7F 01
STB R6, 1000	0000000A 00 17 F0 03
LDI R7, 1000	0000000B 00 17 F0 03
ADD R7, R0, R1	0000000C E2 30 00 00
LDI R8, 1000	0000000D 00 17 F0 03
ADD R8, R0, R1	0000000E E2 30 00 00
LDI R9, 1000	0000000F 00 17 F0 03
ADD R9, R0, R1	00000010 E2 30 00 00
LDI R10, 1000	00000011 00 17 F0 03
ADD R10, R0, R1	00000012 00 17 F0 03
HALT	00000013 00 00 00 00
BREAK	00000014 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.
- 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

The screenshot shows the WeMIPS IDE interface. At the top, there are tabs for 'Run', '32bit', 'Data', and 'ShowWide Device'. Below these are navigation links: 'Addition Doubler', 'Stax', 'Looper', 'Stack Test', and 'Hello World'. Further down are links 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
```

Below the assembly code is a debugger interface with tabs for 'Step' and 'Run'. The 'Run' tab is selected. A checkbox labeled 'Enable auto switching' is checked. The debugger table has columns for Step, T, A, V, Stack, and Log. The data is as follows:

Step	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 are tabs for "ShowHide 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", "Decimal Binary", and "Debug". The main area contains assembly code with comments and memory dump sections. To the right is a register table with columns for \$, T, A, V, Stack, and Log.

\$	T	A	V	Stack	Log
x0				10	
x1				9	
x2				8	
x3				22	
x4				000	
x5				019	
x6				027	
x7				41	

```
# Shows "Hello, world!" at the top of the stack
1.  # .text
2.  .globl _start
3.  .data
4.  _start:    li $t0, 8192
5.  li $t1, 111901
6.  li $t2, 111901
7.  li $t3, 111901
8.  li $t4, 111901
9.  li $t5, 111901
10. li $t6, 111901
11. li $t7, 111901
12. li $t8, 111901
13. li $t9, 111901
14. li $t10, 111901
15. li $t11, 111901
16. li $t12, 111901
17. li $t13, 111901
18. li $t14, 111901
19. li $t15, 111901
20. li $t16, 111901
21. li $t17, 111901
22. li $t18, 111901
23. li $t19, 111901
24. li $t20, 10240000
25. li $t21, 10240000
26. li $t22, 111901
27. li $t23, 0 # (null)
28. li $t24, 12192000
29. li $t25, 4 # 4 is for print string
30. li $t26, 8492
31. # point to the log
32. _symbol:
```

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

MIPS Commands

The screenshot shows a MIPS assembly debugger 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", "Max", "Min", "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 displays the following assembly code:

```
1 # Ensure '$main_stack' is at the top of the stack
2
3 # SB $t0, $t1($sp), $t1 # v
4 # SB $t1, $t1($sp)
5 # SB $t0, $t1($sp), $t1 # v
6 # SB $t1, $t1($sp)
7 # SB $t0, $t1($sp), $t1 # v
8 # SB $t1, $t1($sp)
9 # SB $t0, $t1($sp), $t1 # v
10 # SB $t1, $t1($sp)
11 # ADDI $t0, $t0, 100 # (spnew)
12 # ADDI $t0, $t0, 100 # (spnew)
13 # ADDI $t0, $t0, 100 # (spnew)
14 # ADDI $t0, $t0, 100 # (spnew)
15 # ADDI $t0, $t0, 100 # (spnew)
16 # ADDI $t0, $t0, 100 # (spnew)
17 # ADDI $t0, $t0, 100 # (spnew)
18 # ADDI $t0, $t0, 100 # (spnew)
19 # ADDI $t0, $t0, 100 # (spnew)
20 # ADDI $t0, $t0, 100 # (spnew)
21 # ADDI $t0, $t0, 100 # (spnew)
22 # SB $t0, $t1($sp), $t1 # v
23 # SB $t1, $t1($sp)
24 # SB $t0, $t1($sp), $t1 # v
25 # SB $t1, $t1($sp)
26 # SB $t0, $t1($sp), $t1 # v
27 # SB $t1, $t1($sp)
28 # SB $t0, $t1($sp), $t1 # v
29 # SB $t1, $t1($sp)
30 # SB $t0, $t1($sp), $t1 # v
31 # ADDI $t0, $t0, 4 # $t1 is for print string
32 # ADDI $t0, $t0, 0 # result
33 # System call
```

Below the code, there is a register dump table with columns for S, T, V, Stack, and Log. The registers shown are \$t0 through \$t1, with their values listed in the corresponding columns.

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

MIPS Commands

The screenshot shows a MIPS assembly code editor. At the top, there are tabs for 'ShowHide Demo' and 'User Guide | Unit Tests | Docs'. Below that is a navigation bar with links: 'Addition', 'Subtraction', 'Max', 'Looper', 'Stack Test', 'Hello World', 'Code Gen', 'Save String', 'Interactive', 'Decimal Decimal', and 'Decimal Binary'. A 'Debug' button is also present. The main area contains the following assembly code:

```
1 # Change "Hello world!" at the top of the stack
2
3 .data
4 .text
5
6 li $t0, 11110011          # 111 # e
7 sb $t0, 111000             # 0000000000000000, 111 # e
8
9 li $t1, 00001100           # 0000000000000001, 000 # 1
10 sb $t1, 110100             # 0000000000000000, 000 # 1
11
12 li $t2, 00001110           # 0000000000000011, 000 # e
13 sb $t2, 110010             # 0000000000000000, 000 # e
14
15 li $t3, 00001111           # 0000000000000011, 111 # e
16 add $t0, $t1, $t3            # 1110000000000000, 111 # e
17 li $t4, 00001110           # 0000000000000011, 111 # e
18 add $t0, $t4, $t3            # 1110000000000000, 111 # e
19
20 li $t5, 00001111           # 0000000000000011, 111 # e
21 add $t0, $t5, $t3            # 1110000000000000, 111 # e
22 li $t6, 00001100           # 0000000000000001, 000 # 1
23 add $t0, $t6, $t3            # 1110000000000000, 000 # 1
24 li $t7, 1010000000000000           # 0000000000000000, 100 # e
25
26 li $t8, 1100000000000000           # 0000000000000000, 110 # e
27
28 li $t9, 0000000000000000           # 0000000000000000, 000 # null
29
30 li $t10, 0000000000000000           # 0000000000000000, 000 # null
31 add $t0, $t10, $t3            # 1110000000000000, 000 # null
32
33 add $t0, $t10, $t3            # 1110000000000000, 000 # null
34
35 add $t0, $t10, $t3            # 1110000000000000, 000 # null
36
37 add $t0, $t10, $t3            # 1110000000000000, 000 # null
38
39 add $t0, $t10, $t3            # 1110000000000000, 000 # null
40
41 add $t0, $t10, $t3            # 1110000000000000, 000 # null
42
43 add $t0, $t10, $t3            # 1110000000000000, 000 # null
44
45 add $t0, $t10, $t3            # 1110000000000000, 000 # null
46
47 add $t0, $t10, $t3            # 1110000000000000, 000 # null
48
49 add $t0, $t10, $t3            # 1110000000000000, 000 # null
50
51 add $t0, $t10, $t3            # 1110000000000000, 000 # null
52
53 add $t0, $t10, $t3            # 1110000000000000, 000 # null
54
55 add $t0, $t10, $t3            # 1110000000000000, 000 # null
56
57 add $t0, $t10, $t3            # 1110000000000000, 000 # null
58
59 add $t0, $t10, $t3            # 1110000000000000, 000 # null
59
```

On the right side of the editor, there is a register viewer with columns for \$, T, V, Stack, and Log. The registers show the following values:

Register	\$	T	V	Stack	Log
\$0	10				
\$1	9				
\$2	8				
\$3	22				
\$4	000				
\$5	019				
\$6	027				
\$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:

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', 'Decimal Binary', and 'Debug'. Below the tabs is a 'Step' button and a 'Run' button with a checkbox for 'Create auto stepping'. A status bar at the bottom shows navigation icons.

The assembly code area contains the following MIPS assembly code:

```
1 # Ensure '$t0,$t1,$t2,$t3' are at the top of the stack
2 # SB $t0, $t1($sp), $t2($sp), $t3($sp)
3 # SB $t0, $t1($sp), $t2($sp), $t3($sp) # 4
4 # SB $t0, $t1($sp), $t2($sp), $t3($sp) # 5
5 # SB $t0, $t1($sp), $t2($sp), $t3($sp) # 6
6 # SB $t0, $t1($sp), $t2($sp), $t3($sp) # 7
7 # SB $t0, $t1($sp), $t2($sp), $t3($sp) # 8
8 # SB $t0, $t1($sp), $t2($sp), $t3($sp) # 9
9 # SB $t0, $t1($sp), $t2($sp), $t3($sp) # 10
10 # SB $t0, $t1($sp), $t2($sp), $t3($sp) # 11
11 # SB $t0, $t1($sp), $t2($sp), $t3($sp) # 12
12 # ADDI $t0, $t0, $t0, $t0 # (expand)
13 # ADDI $t0, $t0, $t0, $t0 # 13
14 # ADDI $t0, $t0, $t0, $t0 # 14
15 # ADDI $t0, $t0, $t0, $t0 # 15
16 # ADDI $t0, $t0, $t0, $t0 # 16
17 # ADDI $t0, $t0, $t0, $t0 # 17
18 # ADDI $t0, $t0, $t0, $t0 # 18
19 # ADDI $t0, $t0, $t0, $t0 # 19
20 # ADDI $t0, $t0, $t0, $t0 # 20
21 # ADDI $t0, $t0, $t0, $t0 # 21
22 # ADDI $t0, $t0, $t0, $t0 # 22
23 # ADDI $t0, $t0, $t0, $t0 # 23
24 # ADDI $t0, $t0, $t0, $t0 # 24
25 # ADDI $t0, $t0, $t0, $t0 # 25
26 # ADDI $t0, $t0, $t0, $t0 # 26
27 # ADDI $t0, $t0, $t0, $t0 # 27
28 # ADDI $t0, $t0, $t0, $t0 # 28
29 # ADDI $t0, $t0, $t0, $t0 # 29
30 # ADDI $t0, $t0, $t0, $t0 # 30
31 # ADDI $t0, $t0, $t0, $t0 # 31
32 # ADDI $t0, $t0, $t0, $t0 # 32
33 # ADDI $t0, $t0, $t0, $t0 # 33
34 # ADDI $t0, $t0, $t0, $t0 # 34
35 # ADDI $t0, $t0, $t0, $t0 # 35
36 # ADDI $t0, $t0, $t0, $t0 # 36
37 # ADDI $t0, $t0, $t0, $t0 # 37
38 # ADDI $t0, $t0, $t0, $t0 # 38
39 # ADDI $t0, $t0, $t0, $t0 # 39
40 # ADDI $t0, $t0, $t0, $t0 # 40
41 # ADDI $t0, $t0, $t0, $t0 # 41
```

The register and stack view shows the following values:

	S	T	V	Stack	Log
x0				10	
x1				9	
x2				8	
x3				22	
x4				000	
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

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 tabs is a toolbar with 'Step', 'Run', 'Break', and 'Create auto watching' buttons. A status bar at the bottom shows assembly, binary, hex, and ASCII representations of memory.

The assembly code window contains the following MIPS assembly code:

```
1 # Shows 'Hello, world!' at the top of the stack
2 .text
3 .globl _start
4 .data
5 _str: .asciz "Hello, world!\n"
6 _len: .word 11
7 _buf: .space 100
8 _size: .word 100
9 _str: .asciz "Hello, world!\n"
10 _len: .word 11
11 _buf: .space 100
12 _size: .word 100
13 .text
14 _start: .text
15 _start: .asciz "Hello, world!\n"
16 _start: .size 12, _start, 12P # (expand)
17 _start: .asciz "Hello, world!\n"
18 _start: .size 12, _start, 12P # (expand)
19 _start: .asciz "Hello, world!\n"
20 _start: .size 12, _start, 12P # (expand)
21 _start: .asciz "Hello, world!\n"
22 _start: .size 12, _start, 12P # (expand)
23 _start: .asciz "Hello, world!\n"
24 _start: .size 12, _start, 12P # (expand)
25 _start: .asciz "Hello, world!\n"
26 _start: .size 12, _start, 12P # (expand)
27 _start: .asciz "Hello, world!\n"
28 _start: .size 12, _start, 12P # (expand)
29 _start: .asciz "Hello, world!\n"
30 _start: .size 12, _start, 12P # (expand)
31 _start: .asciz "Hello, world!\n"
32 _start: .size 12, _start, 12P # (expand)
33 _start: .asciz "Hello, world!\n"
34 _start: .size 12, _start, 12P # (expand)
35 _start: .asciz "Hello, world!\n"
36 _start: .size 12, _start, 12P # (expand)
37 _start: .asciz "Hello, world!\n"
38 _start: .size 12, _start, 12P # (expand)
39 _start: .asciz "Hello, world!\n"
40 _start: .size 12, _start, 12P # (expand)
41 _start: .asciz "Hello, world!\n"
42 _start: .size 12, _start, 12P # (expand)
```

The register window on the right shows the following values:

Reg	Value
\$0	10
\$1	9
\$2	8
\$3	22
\$4	000
\$5	019
\$6	027
\$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 the MARS MIPS simulator interface. At the top, there are tabs for "ShowHide Demo", "User Guide", "Unit Tests", and "Docs". Below that is a menu bar with "File", "Edit", and "Run". The main area has tabs for "Assembly", "Datasets", "Hex", "Looper", "Stack View", and "Hello World". Under "Assembly", there are sub-tabs: "Code", "Gen Data String", "Interactive", "Decimal Decimal", and "Decimal Binary". A "Debug" button is also present. The assembly code window contains the following instructions:

```
1 # Shows "Hello world!" at the top of the stack
2
3 # $t0 = 41($sp), $t1 = 42($sp)
4 # $s0 = 43($sp), $s1 = 44($sp)
5 # $s2 = 45($sp), $s3 = 46($sp)
6 # $s4 = 47($sp), $s5 = 48($sp)
7 # $s6 = 49($sp), $s7 = 50($sp)
8 # $s8 = 51($sp), $s9 = 52($sp)
9 # $s10 = 53($sp), $s11 = 54($sp)
10 # $s12 = 55($sp), $s13 = 56($sp)
11 # $s14 = 57($sp), $s15 = 58($sp)
12 # $s16 = 59($sp), $s17 = 60($sp)
13 # $s18 = 61($sp), $s19 = 62($sp) # (sp++)
14 # $s20 = 63($sp), $s21 = 64($sp)
15 # $s22 = 65($sp), $s23 = 66($sp)
16 # $s24 = 67($sp), $s25 = 68($sp)
17 # $s26 = 69($sp), $s27 = 70($sp)
18 # $s28 = 71($sp), $s29 = 72($sp)
19 # $s30 = 73($sp), $s31 = 74($sp)
20 # ADDI $t0,$zero,100 # r1
21 # ADDI $t1,$zero,101 # r2
22 # ADDI $t2,$zero,102 # r3
23 # ADDI $t3,$zero,103 # r4
24 # ADDI $t4,$zero,104 # r5
25 # ADDI $t5,$zero,105 # r6
26 # ADDI $t6,$zero,106 # r7
27 # ADDI $t7,$zero,107 # r8
28 # ADDI $t8,$zero,108 # r9
29 # ADDI $t9,$zero,109 # r10
30 # ADDI $t10,$zero,110 # r11
31 # ADDI $t11,$zero,111 # r12
32 # ADDI $t12,$zero,112 # r13
33 # ADDI $t13,$zero,113 # r14
34 # ADDI $t14,$zero,114 # r15
35 # ADDI $t15,$zero,115 # r16
36 # ADDI $t16,$zero,116 # r17
37 # ADDI $t17,$zero,117 # r18
38 # ADDI $t18,$zero,118 # r19
39 # ADDI $t19,$zero,119 # r20
40 # ADDI $t20,$zero,120 # r21
41 # ADDI $t21,$zero,121 # r22
42 # ADDI $t22,$zero,122 # r23
43 # ADDI $t23,$zero,123 # r24
44 # ADDI $t24,$zero,124 # r25
45 # ADDI $t25,$zero,125 # r26
46 # ADDI $t26,$zero,126 # r27
47 # ADDI $t27,$zero,127 # r28
48 # ADDI $t28,$zero,128 # r29
49 # ADDI $t29,$zero,129 # r30
50 # ADDI $t30,$zero,130 # r31
51 # $t31 = 131($sp) - $t0 = 41($sp)
```

The registers window shows the following values:

x0	10
x1	9
x2	8
x3	22
x4	606
x5	819
x6	927
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 the MARS MIPS simulator interface. The assembly code window displays the following instructions:

```
1 # chores 'Hello, world!' at the top of the stack
2 .text
3 .globl _start
4 .data
5 .word 11110000, 1111 # 0
6 .word 11110000, 1101 # 1
7 .word 11110000, 1001 # 2
8 .word 11110000, 1000 # 3
9 .word 11110000, 1000 # 4
10 .word 11110000, 1100 # 5
11 .word 11110000, 1111 # 6
12 .word 11110000, 1111 # 7 (spnow)
13 .word 11110000, 1100, 1111 # 8
14 .word 11110000, 1100, 1100 # 9
15 .word 11110000, 1100, 1101 # 10
16 .word 11110000, 1100, 1110 # 11
17 .word 11110000, 1100, 1111 # 12
18 .word 11110000, 1100, 1111 # 13
19 .word 11110000, 1100, 1111 # 14
20 .word 11110000, 1100, 1111 # 15
21 .word 11110000, 1100, 1111 # 16
22 .word 11110000, 1100, 1111 # 17
23 .word 11110000, 1100, 1100 # 18
24 .word 11110000, 1100, 1100 # 19
25 .word 11110000, 1100, 1101 # 20
26 .word 11110000, 1100, 1101 # 21
27 .word 11110000, 1100, 1101 # 22
28 .word 11110000, 1100, 1101 # 23
29 .word 11110000, 1100, 1101 # 24
30 .word 11110000, 1100, 1101 # 25
31 .word 11110000, 1100, 1101 # 26
32 .word 11110000, 1100, 1101 # 27
33 .word 11110000, 1100, 1101 # 28
34 .word 11110000, 1100, 1101 # 29
35 .word 11110000, 1100, 1101 # 30
36 .word 11110000, 1100, 1101 # 31
```

The registers window shows the following values:

Reg	Value
\$0	10
\$1	9
\$2	8
\$3	22
\$4	0000
\$5	819
\$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.
addi \$s1, \$s2, 100 (Basic form: OP rd, rs, imm)
- **J Instructions:** instructions that jump to another memory location.

MIPS Commands

The screenshot shows a MIPS assembly debugger interface. The assembly code window contains the following instructions:

```
1 # Shows 'Hello, world!' at the top of the stack
2 .text
3 .globl _start
4 .type _start, @function
5 _start:
6    li $t0, 115902
7    li $t1, 115901
8    li $t2, 115900
9    li $t3, 115901
10   li $t4, 115900
11   li $t5, 115901
12   li $t6, 115900
13   li $t7, 115901
14   li $t8, 115900
15   li $t9, 115901
16   li $t10, 115900
17   li $t11, 115901
18   li $t12, 115900
19   addi $t0, $t0, 1000000000
20   addi $t1, $t1, 1000000000
21   addi $t2, $t2, 1000000000
22   addi $t3, $t3, 1000000000
23   addi $t4, $t4, 1000000000
24   addi $t5, $t5, 1000000000
25   addi $t6, $t6, 1000000000
26   addi $t7, $t7, 1000000000
27   addi $t8, $t8, 1000000000
28   addi $t9, $t9, 1000000000
29   addi $t10, $t10, 1000000000
30   addi $t11, $t11, 1000000000
31   addi $t12, $t12, 1000000000
32   addi $t0, $t0, 4 # $t0 is for print string
33   addi $t0, $t0, 0 # point to the log
34 _symbol:
```

The register window shows the following values:

Reg	Value
\$t0	10
\$t1	9
\$t2	8
\$t3	22
\$t4	0000
\$t5	819
\$t6	827
\$t7	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.
j done

MIPS Commands

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
 - **R Instructions:** Commands that use data in the registers:
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

User | 3 | Dat ShowWide Device

Addition Doubler Stax Looper Stack Test Hello World

Code Gen Save String Interactive Binary2 Decimal2 Decimal2 Binary

Debug

```
# Store "Hello world!" at the top of the stack
1 ADDI $t0, $zero, 72 # $H
2 ADDI $t1, $zero, 101 # $e
3 ADDI $t2, $zero, 101 # $n
4 ADDI $t3, $zero, 101 # $o
5 ADDI $t4, $zero, 101 # $l
6 ADDI $t5, $zero, 101 # $w
7 ADDI $t6, $zero, 101 # $r
8 ADDI $t7, $zero, 101 # $d
9 ADDI $t8, $zero, 101 # $c
10 ADDI $t9, $zero, 101 # $o
11 ADDI $t10, $zero, 101 # $n
12 ADDI $t11, $zero, 101 # $o
13 ADDI $t12, $zero, 101 # $e
14 ADDI $t13, $zero, 101 # $l
15 ADDI $t14, $zero, 101 # $d
16 ADDI $t15, $zero, 101 # $c
17 ADDI $t16, $zero, 101 # $o
18 ADDI $t17, $zero, 101 # $l
19 ADDI $t18, $zero, 101 # $w
20 ADDI $t19, $zero, 101 # $r
21 ADDI $t20, $zero, 101 # $c
22 ADDI $t21, $zero, 101 # $l
23 ADDI $t22, $zero, 101 # $o
24 ADDI $t23, $zero, 101 # $d
25 ADDI $t24, $zero, 101 # $c
26 ADDI $t25, $zero, 33 # !
27 ADDI $t26, $zero, 101 # $l
28 ADDI $t27, $zero, 0 # (null)
29 ADDI $t28, $zero, 101 # $l
30 ADDI $t29, $zero, 4 # 4 is for print string
31 ADDI $t30, $zero, 0 # point to the log
32 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:	995				
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.



A screenshot of a debugger interface. On the left is the assembly code window, showing a series of assembly instructions. On the right is the registers window, showing various CPU register values. The assembly code includes labels like `start:`, `loop:`, and `end:`, along with various arithmetic and logical operations.

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.



A screenshot of a hex editor application. The left pane shows a list of memory pages, with the first page selected. The right pane displays assembly code in a text-based interface. The code consists of several lines of assembly instructions, likely a loop, starting with a label and ending with a jump back to the label. The assembly code includes labels like `start:`, `loop:`, and `end:`, and instructions like `add r0, r1, r0` and `jmp loop`.

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: 18 Go!

Show/Hide Demos

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 test when you reach 26
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
11 J SETUP
12 DONE: ADDI $t0, $zero, 0 # Null (0) to terminate string
13 SB $t0, 0($sp)          # Add null to stack
14 ADDI $sp, $sp, -26      # Set up stack to print
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)

Step Run Enable auto switching

S T A V Stack Log

[Clear Log](#)

Emulation complete, returning to line 1

abcdefghijklmnopqrstuvwxyz

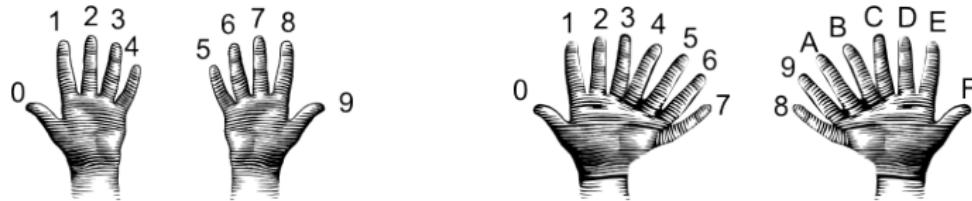


Today's Topics



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

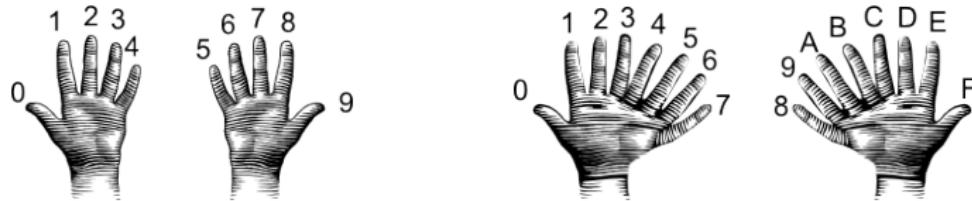
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

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

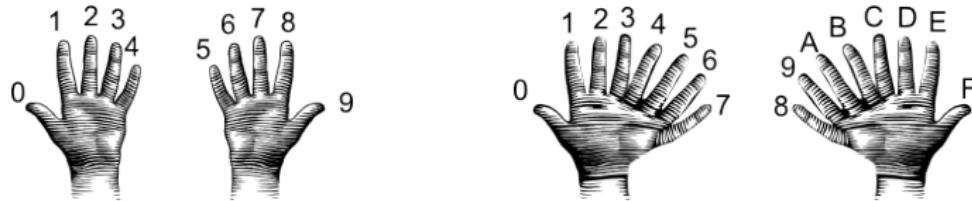
Hexadecimal to Decimal: Converting Between Bases



(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.

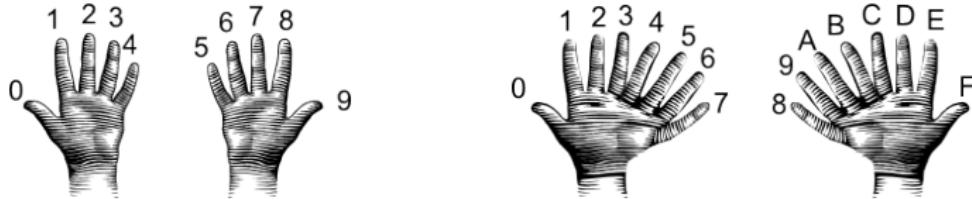
Hexadecimal to Decimal: Converting Between Bases



(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?

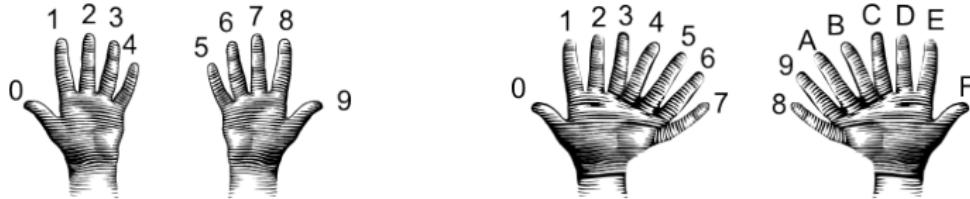
Hexadecimal to Decimal: Converting Between Bases



(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.

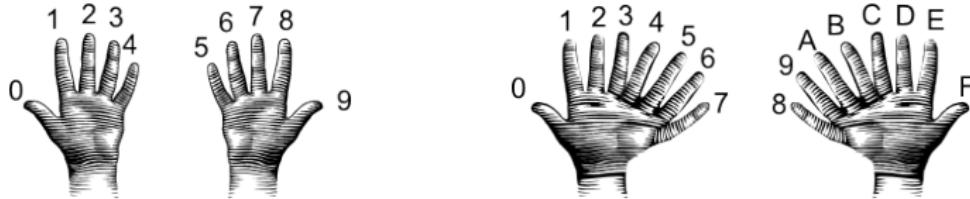
Hexadecimal to Decimal: Converting Between Bases



(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.

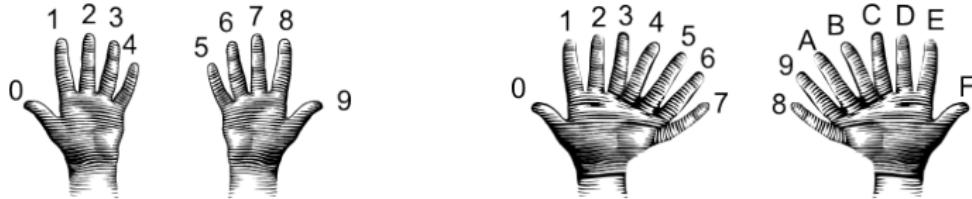
Hexadecimal to Decimal: Converting Between Bases



(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):

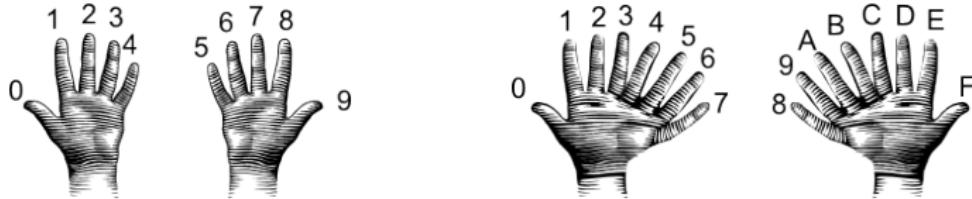
- Convert first digit to decimal and multiple by 16.
 - Convert second digit to decimal and add to total.
 - Example: what is 2A as a decimal number?

2 in decimal is 2. 2×16 is 32.

A in decimal digits is 10.

$32 + 10$ is 42.

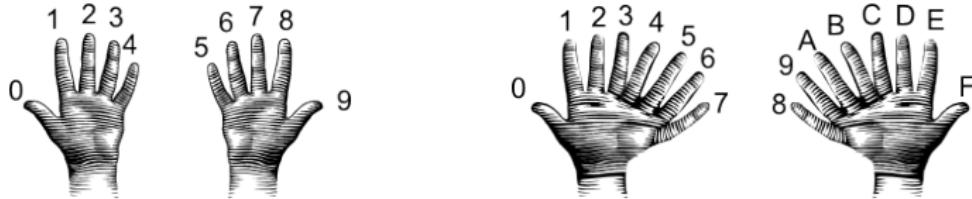
Hexadecimal to Decimal: Converting Between Bases



(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?

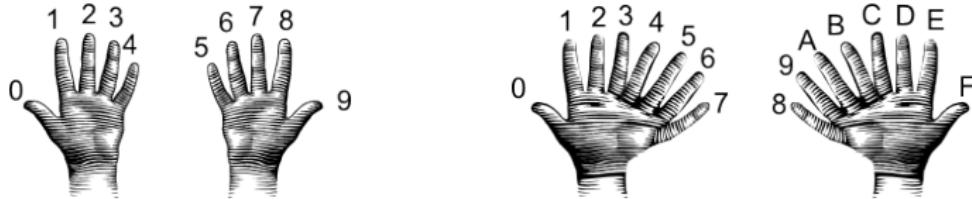
Hexadecimal to Decimal: Converting Between Bases



(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.

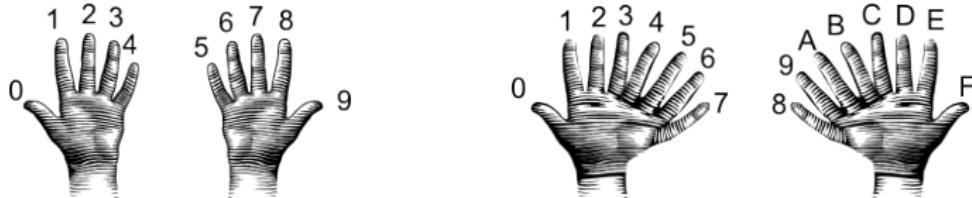
Hexadecimal to Decimal: Converting Between Bases



(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.

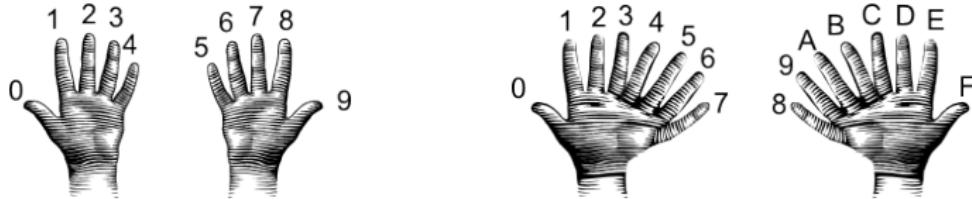
Hexadecimal to Decimal: Converting Between Bases



(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.

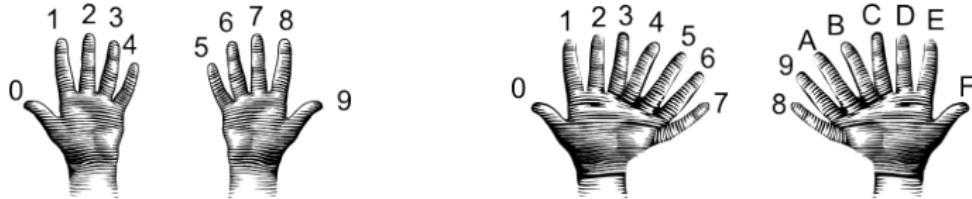
- Example: what is 99 as a decimal number?

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?

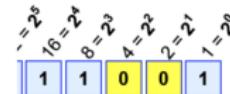
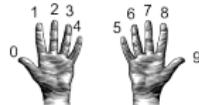
9 in decimal is 9. 9×16 is 144.

9 in decimal digits is 9

$144 + 9$ is 153.

Answer is 153.

Decimal to Binary: Converting Between Bases

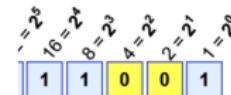
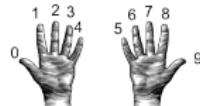


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From decimal to binary:

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

Decimal to Binary: Converting Between Bases

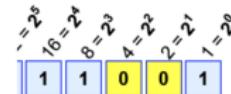
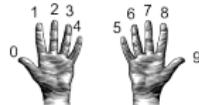


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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.

Decimal to Binary: Converting Between Bases

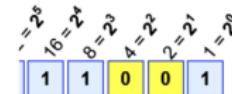


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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.

Decimal to Binary: Converting Between Bases

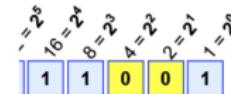


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

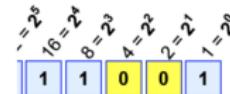
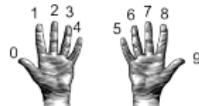


$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- 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

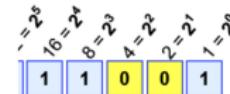
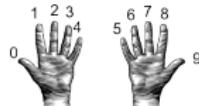


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

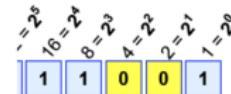
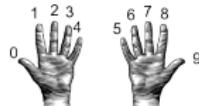


$$\text{Example: } 1 \times 16 + 1 \times 8 + 0 \times 4 + 1 \times 1 = 16 + 8 + 1 = 25$$

- 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.
- 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

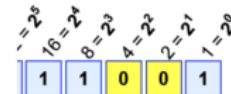
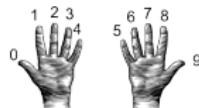


$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- 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.

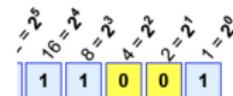
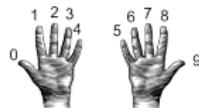
Decimal to Binary: Converting Between Bases



- 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



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

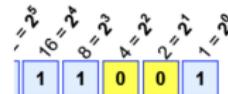
- From decimal to binary:

- 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



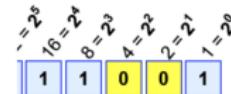
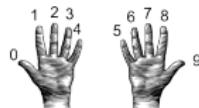
- From decimal to binary:

- 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



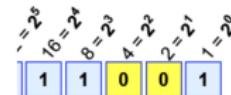
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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.

Decimal to Binary: Converting Between Bases



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

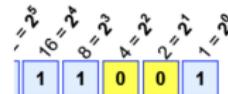
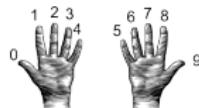
- From decimal to binary:

- 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



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

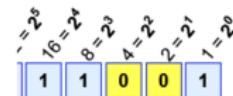
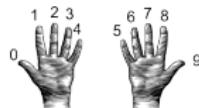
- From decimal to binary:

- 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:

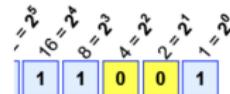
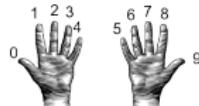
- Divide by $128 (= 2^7)$. Quotient is the first digit.
- Divide remainder by $64 (= 2^6)$. Quotient is the next digit.
- Divide remainder by $32 (= 2^5)$. Quotient is the next digit.
- Divide remainder by $16 (= 2^4)$. Quotient is the next digit.
- Divide remainder by $8 (= 2^3)$. Quotient is the next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

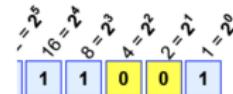
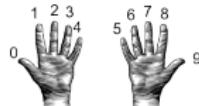
- Divide by $128 (= 2^7)$. Quotient is the first digit.
- Divide remainder by $64 (= 2^6)$. Quotient is the next digit.
- Divide remainder by $32 (= 2^5)$. Quotient is the next digit.
- Divide remainder by $16 (= 2^4)$. Quotient is the next digit.
- Divide remainder by $8 (= 2^3)$. Quotient is the next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

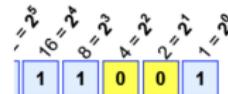
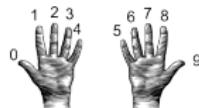
- Divide by $128 (= 2^7)$. Quotient is the first digit.
- Divide remainder by $64 (= 2^6)$. Quotient is the next digit.
- Divide remainder by $32 (= 2^5)$. Quotient is the next digit.
- Divide remainder by $16 (= 2^4)$. Quotient is the next digit.
- Divide remainder by $8 (= 2^3)$. Quotient is the next 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: $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 0 \times 1 + 1 \times 1 = 16 + 8 + 4 + 1 = 25$

- 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.
- 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?

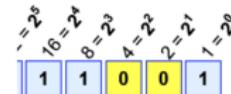
130/128 is 1 rem 2. First digit is 1: 1...

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?

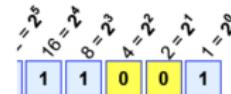
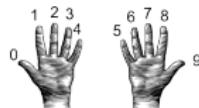
130/128 is 1 rem 2. First digit is 1: 1...

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?

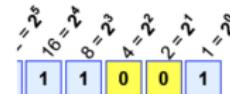
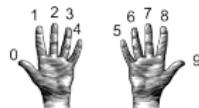
130/128 is 1 rem 2. First digit is 1: 1...

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...

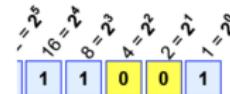
2/64 is 0 rem 2. Next digit is 0: 10...

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

- 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...

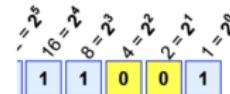
2/64 is 0 rem 2. Next digit is 0: 10...

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:

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...

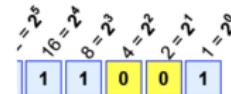
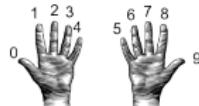
2/64 is 0 rem 2. Next digit is 0: 10...

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



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...

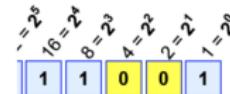
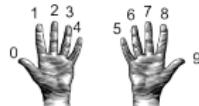
2/32 is 0 rem 2. Next digit is 0: 100...

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.

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...

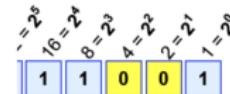
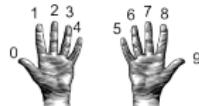
2/32 is 0 rem 2. Next digit is 0: 100...

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...

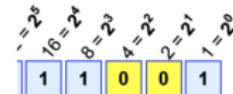
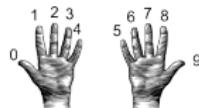
2/32 is 0 rem 2. Next digit is 0: 100...

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: $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 = 25$

- 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...

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

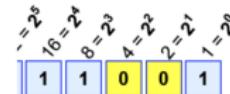
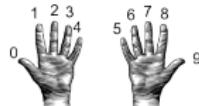
2/16 is 0 rem 2. Next digit is 0: 1000...

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...

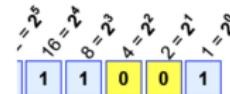
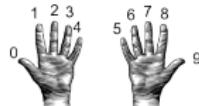
2/16 is 0 rem 2. Next digit is 0: 1000...

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...

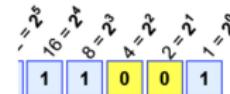
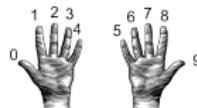
2/16 is 0 rem 2. Next digit is 0: 1000...

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...

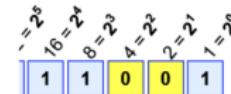
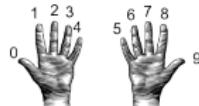
2/8 is 0 rem 2. Next digit is 0: 10000...

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



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...

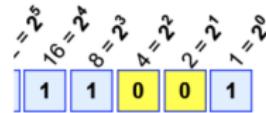
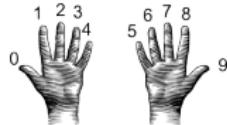
2/8 is 0 rem 2. Next digit is 0: 10000...

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

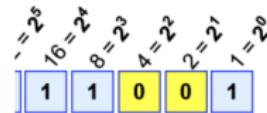
Decimal to Binary: Converting Between Bases



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

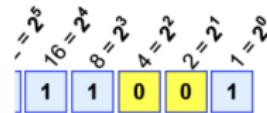


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

$99 / 128$ is 0 rem 99.

Decimal to Binary: Converting Between Bases

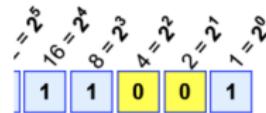


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

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

Decimal to Binary: Converting Between Bases



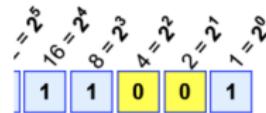
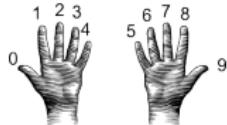
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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



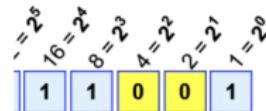
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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



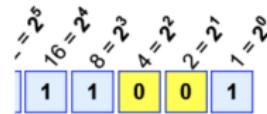
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

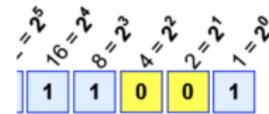
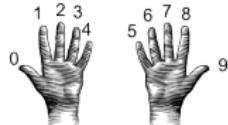
- Example: what is 99 in binary notation?

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.

Decimal to Binary: Converting Between Bases



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

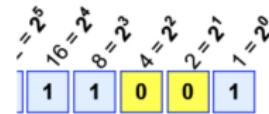
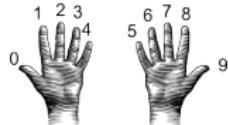
- Example: what is 99 in binary notation?

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

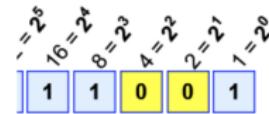
- Example: what is 99 in binary notation?

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

- Example: what is 99 in binary notation?

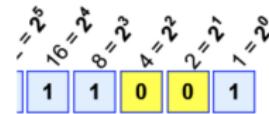
99/128 is 0 rem 99. First digit is 0: 0...

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.

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?

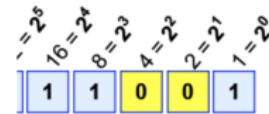
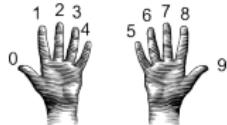
99/128 is 0 rem 99. First digit is 0: 0...

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?

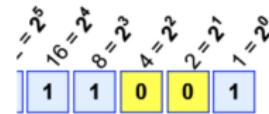
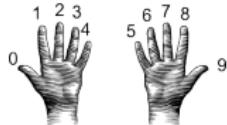
99/128 is 0 rem 99. First digit is 0: 0...

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...

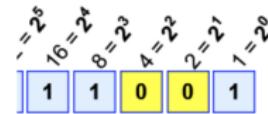
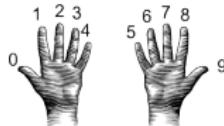
99/64 is 1 rem 35. Next digit is 1: 01...

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...

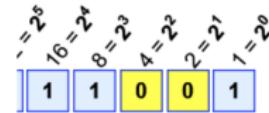
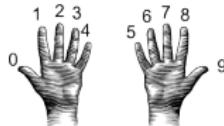
99/64 is 1 rem 35. Next digit is 1: 01...

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...

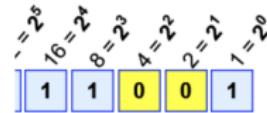
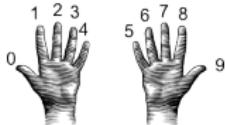
99/64 is 1 rem 35. Next digit is 1: 01...

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...

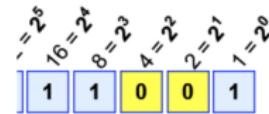
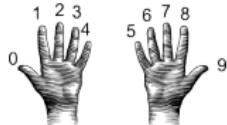
35/32 is 1 rem 3. Next digit is 1: 011...

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...

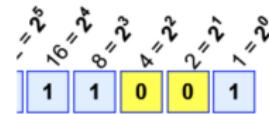
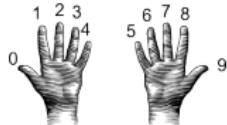
35/32 is 1 rem 3. Next digit is 1: 011...

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...

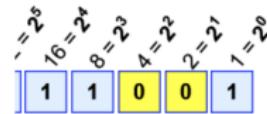
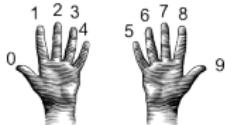
35/32 is 1 rem 3. Next digit is 1: 011...

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...

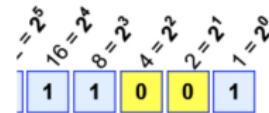
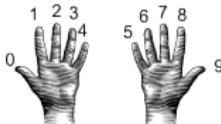
3/16 is 0 rem 3. Next digit is 0: 0110...

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...

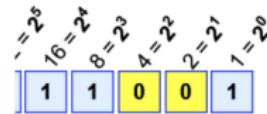
3/16 is 0 rem 3. Next digit is 0: 0110...

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...

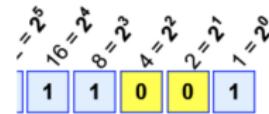
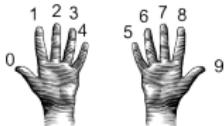
3/16 is 0 rem 3. Next digit is 0: 0110...

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...

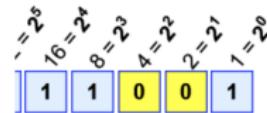
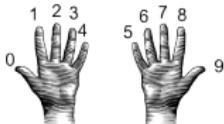
3/8 is 0 rem 3. Next digit is 0: 01100...

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...

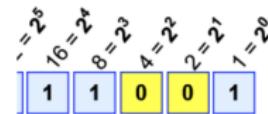
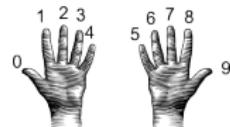
3/4 is 0 remainder 3. Next digit is 0: 011000...

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

Adding the last remainder: 01100011

Answer is 1100011.

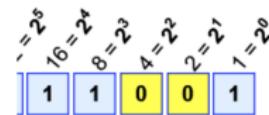
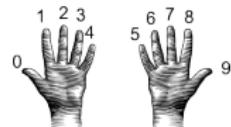
Binary to Decimal: Converting Between Bases



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

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

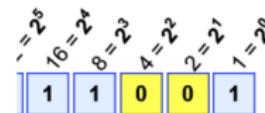
Binary to Decimal: Converting Between Bases



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

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

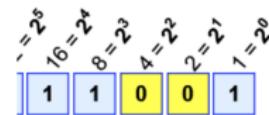
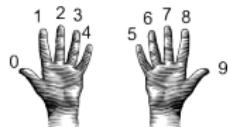
Binary to Decimal: Converting Between Bases



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.

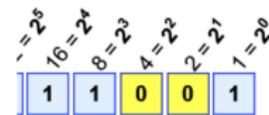
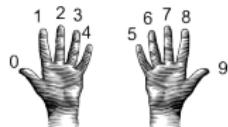
Binary to Decimal: Converting Between Bases



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

- 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

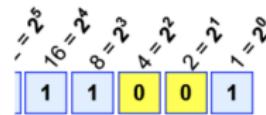
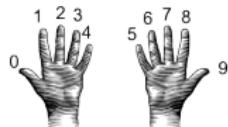


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

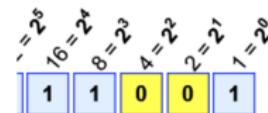
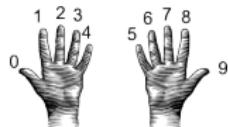


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

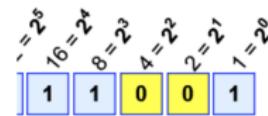
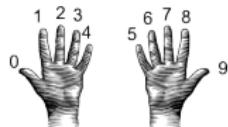


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From binary to decimal:

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

Binary to Decimal: Converting Between Bases

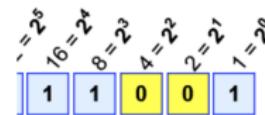
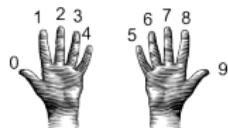


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

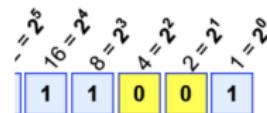
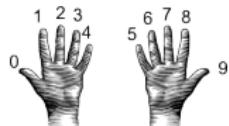


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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.

Binary to Decimal: Converting Between Bases



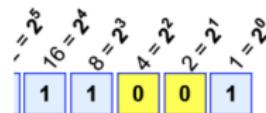
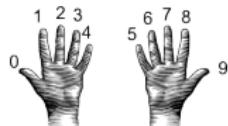
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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:

Binary to Decimal: Converting Between Bases



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

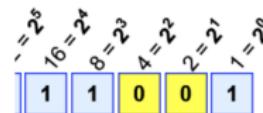
- From binary to decimal:

- 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:

Binary to Decimal: Converting Between Bases



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

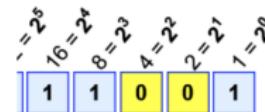
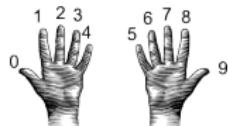
- From binary to decimal:

- 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:

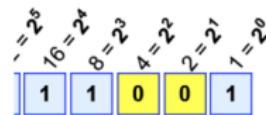
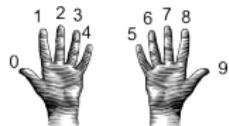
- Set sum = last digit.
- Multiply next digit by 2^1 . Add to sum.
- Multiply next digit by $4 = 2^2$. Add to sum.
- Multiply next digit by $8 = 2^3$. Add to sum.
- Multiply next digit by $16 = 2^4$. Add to sum.
- Multiply next digit by $32 = 2^5$. Add to sum.
- Multiply next digit by $64 = 2^6$. Add 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: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From binary to decimal:

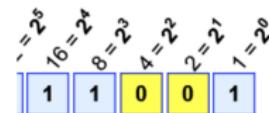
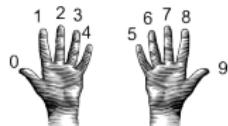
- Set sum = last digit.
- Multiply next digit by $2 = 2^1$. Add to sum.
- Multiply next digit by $4 = 2^2$. Add to sum.
- Multiply next digit by $8 = 2^3$. Add to sum.
- Multiply next digit by $16 = 2^4$. Add to sum.
- Multiply next digit by $32 = 2^5$. Add to sum.
- Multiply next digit by $64 = 2^6$.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

$0 \times 2 = 0$. 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?

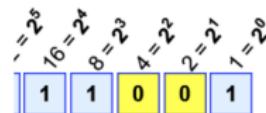
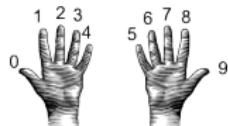
Sum starts with: 1

$0 \times 2 = 0$. 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



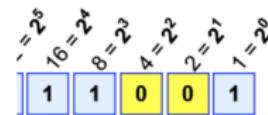
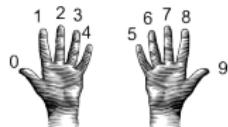
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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
 $0 \times 2 = 0$. Add 0 to sum: 1
 $1 \times 4 = 4$. Add 4 to sum: 5
 $1 \times 8 = 8$. Add 8 to sum: 13

Binary to Decimal: Converting Between Bases



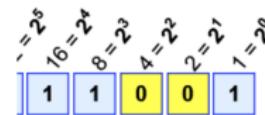
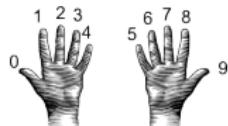
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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:

Binary to Decimal: Converting Between Bases



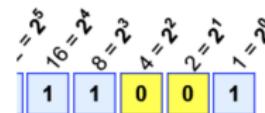
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From binary to decimal:

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

Sum starts with: 1
0*2 = 0. Add 0 to sum: 1
1*4 = 4. Add 4 to sum: 5
1*8 = 8. Add 8 to sum: 13
1*16 = 16. Add 16 to sum: 29

Binary to Decimal: Converting Between Bases



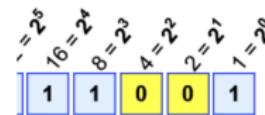
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From binary to decimal:

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

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



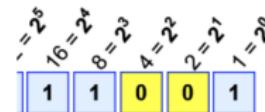
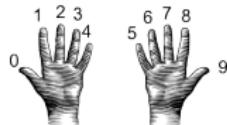
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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



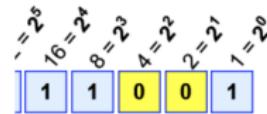
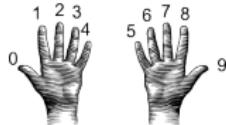
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From binary to decimal:

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

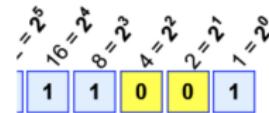


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with:

Binary to Decimal: Converting Between Bases



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

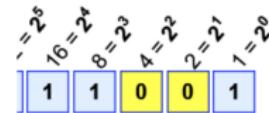
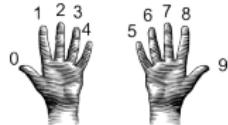
- Example: What is 10100100 in decimal?

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?

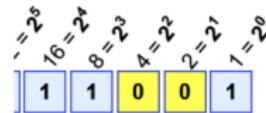
Sum starts with:

0

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

0

Binary to Decimal: Converting Between Bases



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

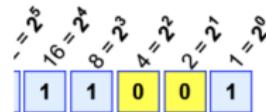
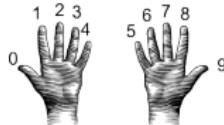
- Example: What is 10100100 in decimal?

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

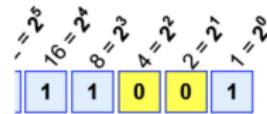
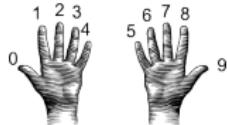
- Example: What is 10100100 in decimal?

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

- Example: What is 10100100 in decimal?

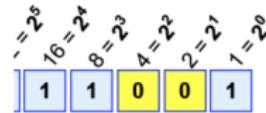
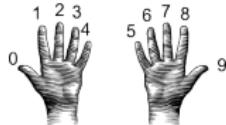
Sum starts with: 0

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

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

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

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?

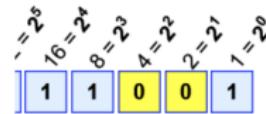
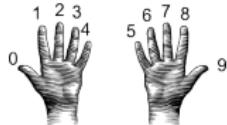
Sum starts with: 0

$0 \times 2 = 0$. 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

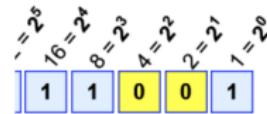
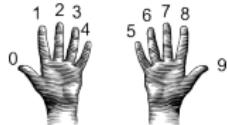
$0 \times 2 = 0$. Add 0 to sum: 0

$1 \times 4 = 4$. 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

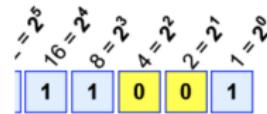
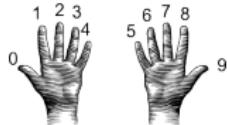
$0 \times 2 = 0$. Add 0 to sum: 0

$1 \times 4 = 4$. 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

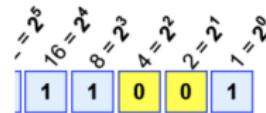
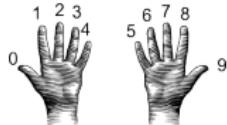
$1 \times 4 = 4$. Add 4 to sum: 4

$0 \times 8 = 0$. 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

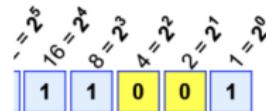
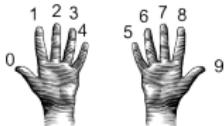


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

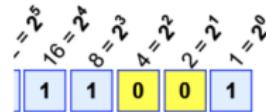
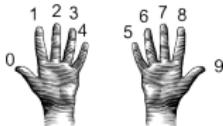
$0 \times 8 = 0$. Add 0 to sum: 4

$0 \times 16 = 0$. 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

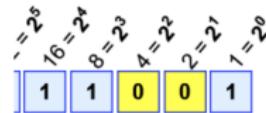
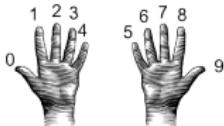
$0 \times 8 = 0$. Add 0 to sum: 4

$0 \times 16 = 0$. 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: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

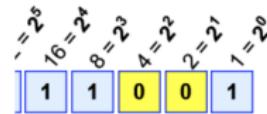
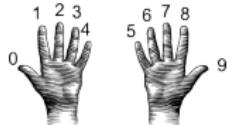
$0 \times 16 = 0$. Add 0 to sum: 4

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

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

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

Binary to Decimal: Converting Between Bases

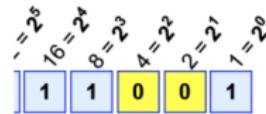
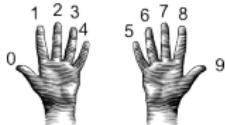


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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



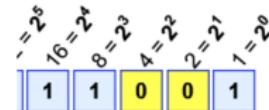
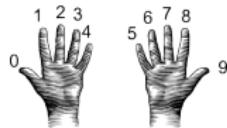
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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.

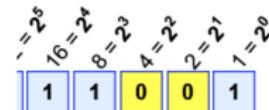
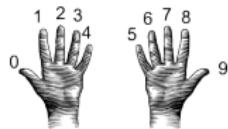
Design Challenge: Incrementers



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

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

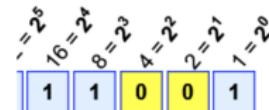
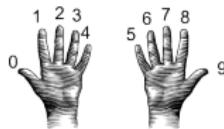
Design Challenge: Incrementers



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

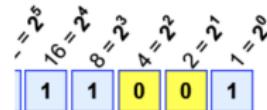
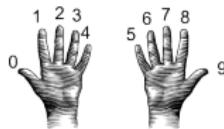


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

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

Design Challenge: Incrementers



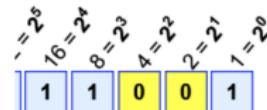
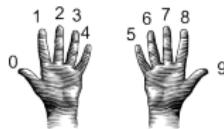
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- 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.

Design Challenge: Incrementers



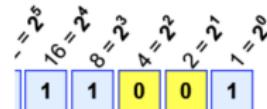
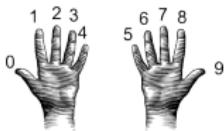
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

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

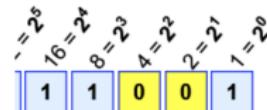
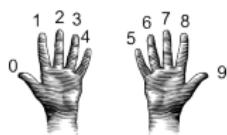
- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

```
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



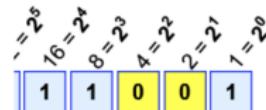
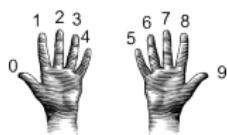
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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



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 on 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

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

18 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 8.5" x 11" piece of paper that you can write on.
- When taking the exam, you may have with you pens and pencils, and your note sheet.
- You may not use a computer, calculator, tablet, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating or communism, among others) as serious violations of the Honor Code. Such acts are considered a violation against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy on Academic Integrity and punishes acts of academic dishonesty according to the Hunter College Academic Integrity Procedures.

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

Exam Options

Exam Times:

- Default Regular Time: Monday, 24 May, 9-11am.

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

18 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 8.5" x 11" piece of paper that you can write on.
- When taking the exam, you may have with you pens and pencils, and your note sheet.
- You may not use a computer, calculator, tablet, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating or communism, among others) as serious violations of the Honor Code. Such acts are considered a violation against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy on Academic Integrity and punishes acts of academic dishonesty according to the Hunter College Academic Integrity Procedure.

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

Exam Options

Exam Times:

- Default Regular Time: Monday, 24 May, 9-11am.
- Alternate Time: Friday, 21 May, 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 8.5" x 11" piece of paper that you can write on.
- When taking the exam, you may have with you pens and pencils, and your note sheet.
- You may not use a computer, calculator, tablet, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating or communism, among others) as serious violations of the Honor Code. Such acts are considered a violation against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy on Academic Integrity and pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedure.

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

Exam Options

Exam Times:

- Default Regular Time: Monday, 24 May, 9-11am.
- Alternate Time: Friday, 21 May, 8am-10am.
- Accessibility Testing: If you have not done so already, email me no later than 7 May.

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 8.5" x 11" piece of paper that you can write on.
- When taking the exam, you may have with you pens and pencils, and your note sheet.
- You may not use a computer, calculator, tablet, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating or communism, among others) as serious violations of the Honor Code. Such acts are considered a violation against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy on Academic Integrity and pursuing cases of academic dishonesty according to the Hunter College Academic Integrity Procedure.

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

Exam Options

Exam Times:

- Default Regular Time: Monday, 24 May, 9-11am.
- Alternate Time: Friday, 21 May, 8am-10am.
- Accessibility Testing: If you have not done so already, email me no later than 7 May.
- Survey for your exam date choice will be available next lecture. **No survey answer implies you will take the exam on 24 May.**

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 8.5" x 11" piece of paper that you can write on.
- When taking the exam, you may have with you pens and pencils, and your note sheet.
- You may not use a computer, calculator, tablet, smart phone, 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, among others) as serious violations of the Honor Code. Such acts will be referred to the Office of Student Conduct and will result in disciplinary action against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy on Academic Integrity and pursues acts of academic dishonesty according to the Hunter College Academic Integrity Procedure.

Indicates that all cases of academic dishonesty will be reported to the Office of Student Conduct and will result in sanctions.
Name: _____
SSN: _____
Email: _____
Signature: _____

Exam Options

Exam Times:

- Default Regular Time: Monday, 24 May, 9-11am.
- Alternate Time: Friday, 21 May, 8am-10am.
- Accessibility Testing: If you have not done so already, email me no later than 7 May.
- Survey for your exam date choice will be available next lecture. **No survey answer implies you will take the exam on 24 May.**
- If you choose to take the early date, **you will not be given access to the exam on 24 May even if you miss the early exam.**

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 8.5" x 11" piece of paper that you can write on.
- When taking the exam, you may bring with you pens and pencils, and your note sheet.
- You may not use a computer, calculator, tablet, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating or communism, among others) as serious violations of the Honor Code. Such acts will be referred to the Office of Student Conduct and will result in sanctions against the violators of individual integrity. The College is committed to enforcing the CUNY Policy on Academic Integrity and the various forms of academic dishonesty according to the Hunter College Academic Integrity Procedures.

I acknowledge that all forms of academic dishonesty will be reported to the Office of Student Conduct and will result in sanctions.
Name: _____
Social ID: _____
Email: _____
Signature: _____

Exam Options

Exam Times:

- Default Regular Time: Monday, 24 May, 9-11am.
- Alternate Time: Friday, 21 May, 8am-10am.
- Accessibility Testing: If you have not done so already, email me no later than 7 May.
- Survey for your exam date choice will be available next lecture. **No survey answer implies you will take the exam on 24 May.**
- If you choose to take the early date, **you will not be given access to the exam on 24 May even if you miss the early exam.**

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 8.5" x 11" piece of paper that you can write on.
- When taking the exam, you may bring with you pens and pencils, and your note sheet.
- You may not use a computer, calculator, tablet, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating or communism, among others) as serious violations of the Honor Code. Such acts will be referred to the Office of Student Conduct and will result in sanctions against the violators of individual integrity. The College is committed to enforcing the CUNY Policy on Academic Integrity and the various forms of academic dishonesty according to the Hunter College Academic Integrity Procedures.

I acknowledge that all forms of academic dishonesty will be reported to the Office of Student Conduct and will result in sanctions.
Name: _____
Social ID: _____
Email: _____
Signature: _____

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 Lab Review (Zoom links on Blackboard / Syncrhonous Meetings)

Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Optional - attend Lab Review (Zoom links on Blackboard / Syncrhonous Meetings)
- 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 Lab Review (Zoom links on Blackboard / Syncrhonous Meetings)
- Take the Lab Quiz on Gradescope by 6pm on Wednesday
- Submit this week's 5 programming assignments (programs 50-52)

Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Optional - attend Lab Review (Zoom links on Blackboard / Syncrhonous Meetings)
- Take the Lab Quiz on Gradescope by 6pm on Wednesday
- Submit this week's 5 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 Lab Review (Zoom links on Blackboard / Syncrhonous Meetings)
- Take the Lab Quiz on Gradescope by 6pm on Wednesday
- Submit this week's 5 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)