Python Data Types

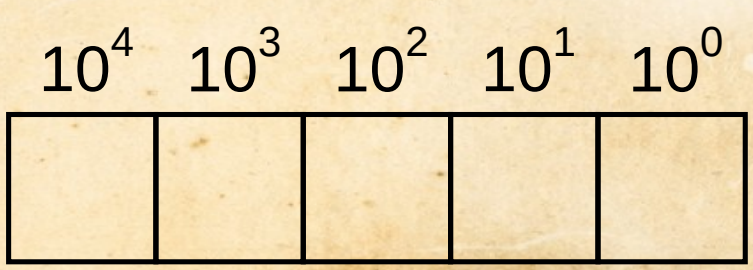
Numbers

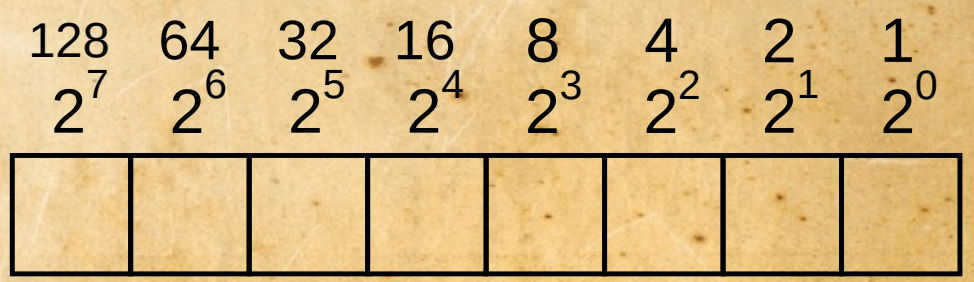
Lecture Contents

- Review of Number Systems
- How Computers Store Data
  - Integers
  - Real Numbers
  - Decimals
  - Fractions

Review of Number Systems

- The ***binary*** system uses ***place value*** and two digits (0, 1) to represent a value.
- Although we could, we generally don't see a decimal point used with binary numbers.
- For denary, each place value is a power of ten. For binary, each place value is a power of two.

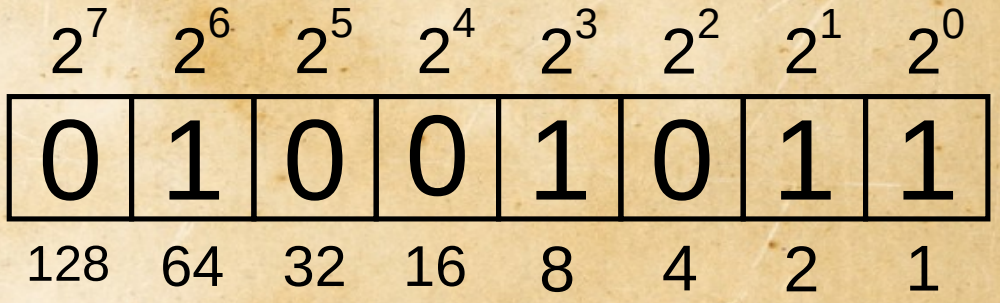




Review of Number Systems

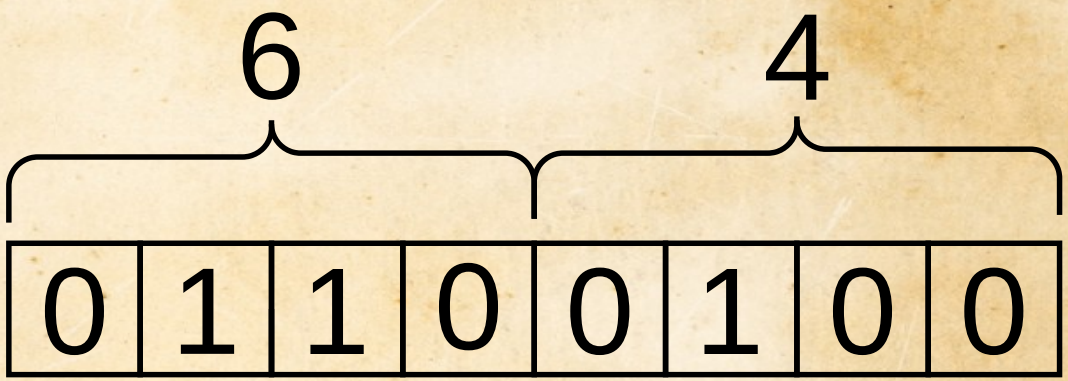
- Be able to convert back and forth between ***binary*** and ***denary***.

75−64 = 11
11−8 = 3
3−2 = 1
1−1 = 0



Review of Number Systems

- Be able to convert back and forth between ***binary*** and ***hexadecimal***.



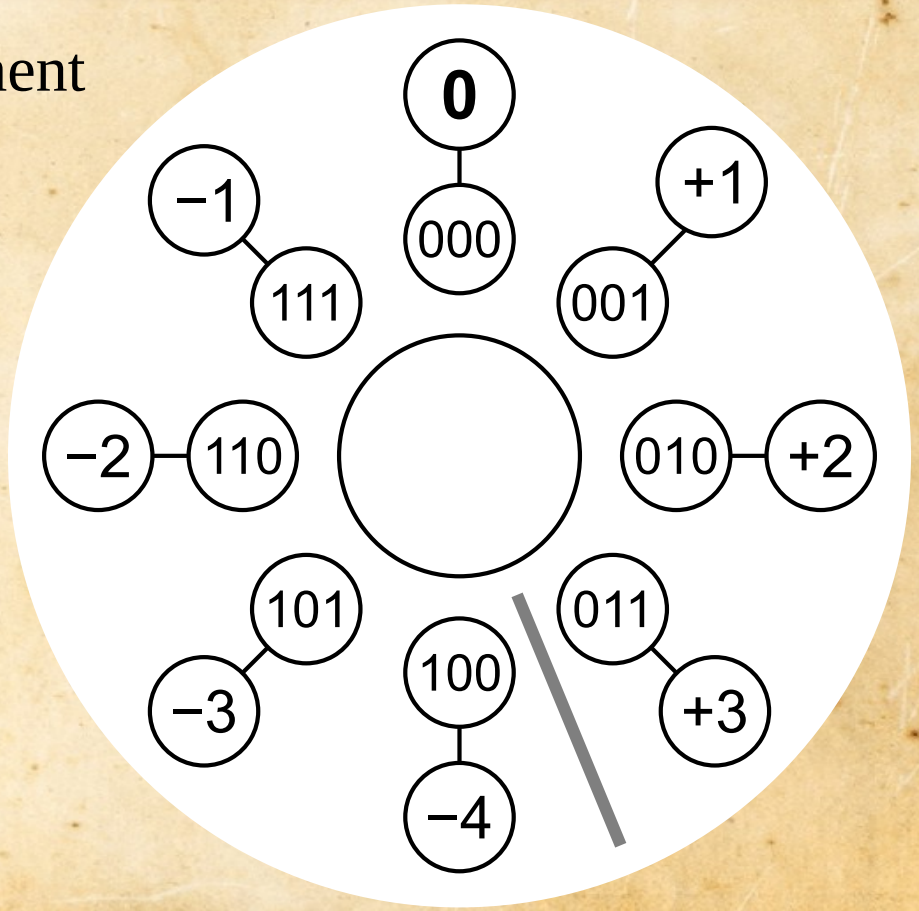
Decimal	Hexa-decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Lecture Contents

- Review of Number Systems
- How Computers Store Data
  - **Integers**
  - Real Numbers
  - Decimals
  - Fractions

How Computers Store Numbers

- Two’s complement

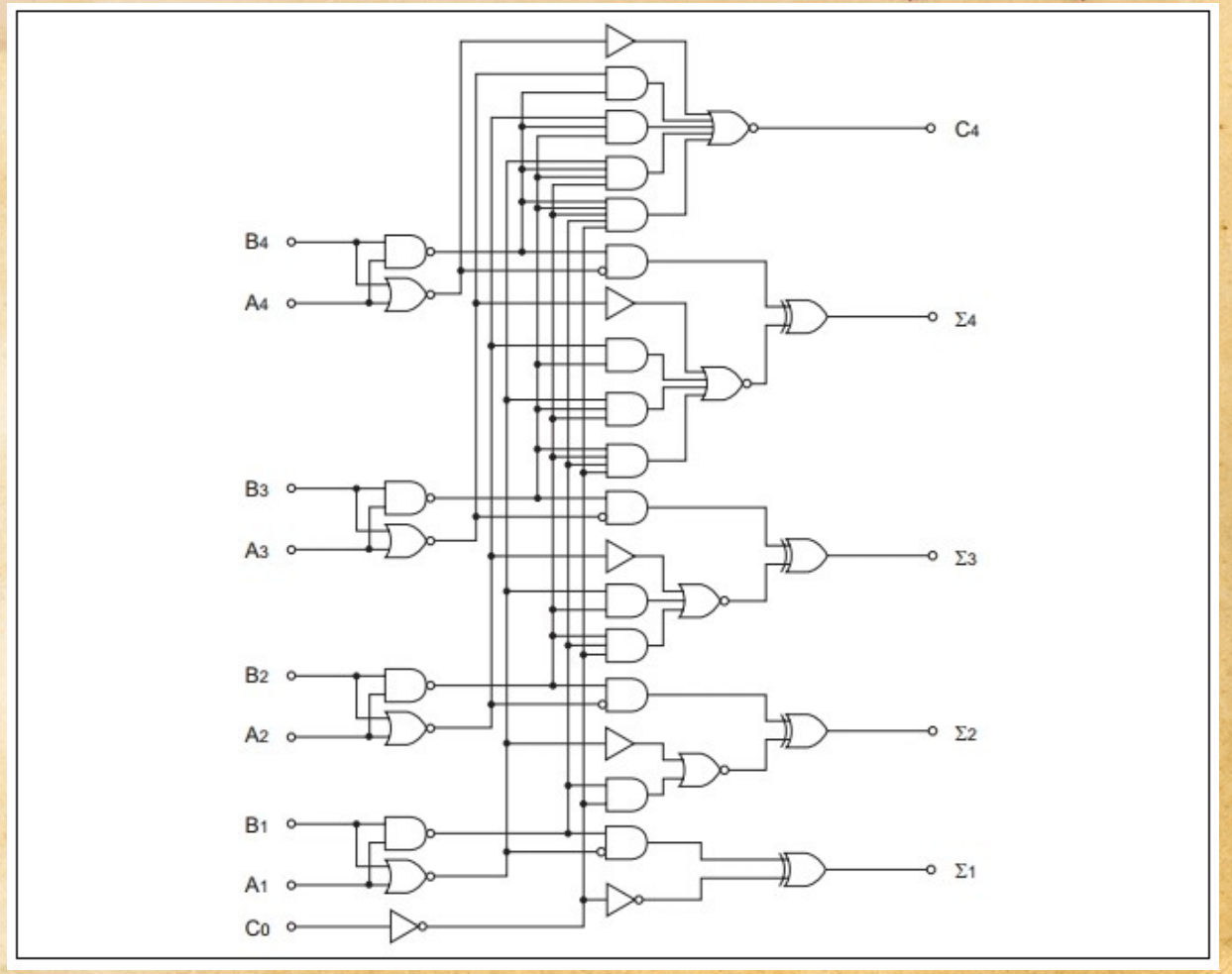


How Computers Store Data

- Integers:
  - almost exclusively stored as either an ***unsigned*** integer or as a ***two's complement*** (signed) number.
  - the hardware to perform an arithmetic operations does not differ between unsigned and signed integers.
  - overflow of unsigned and signed numbers occurs at different values, for example, with 8-bit integers:
    - Unsigned: 255 + 1 = 0 ← overflow for unsigned number!
    - Signed: 127 + 1 = –128 ← overflow for two's complement!
  - Overflow is detected in hardware by many microprocessors (x86, ARM)
    - RISC-V has no integer overflow detection – instructions must be added by compiler / interpreter / assembly programmer.

How Computers Store Data

- Integers
  - 4-bit carry adder
    - The logic that can add two 4-bit binary numbers
    - Σ_1 to Σ_4 contain the addition result
    - C_0 is the carry-in from the previous adder stage
    - C_4 either carries to the next adder, or signals overflow from the most significant adder



How Computers Store Data

- Integers
  - Processors (and most programming languages) have a set number of bits to store an integer value
    - Typically a multiple of 8 bits
    - Modern general-purpose computers: 64 bits
    - Microcontrollers often use 32 bits
    - Many programming languages default to 32 bits (e.g.: Java `int`)

List of Java Integer Types

- *Integers*
  - **`byte`** (8 bits)
  - **`short`** (16 bits)
  - **`int`** (32 bits)
  - **`long`** (64 bits)

Java has no unsigned numbers, unlike most other programming languages.

How Computers Store Data

- Python integers – `int` class
  - Dynamically expands
    - If there's overflow with an operation, more bits are allocated to the integer storage!
    - Upper value limited only by memory space available
  - If you have extremely large integers, operations will take more time to execute
    - Microprocessor hardware still must operate on 64-bit values (or shorter)

How Computers Store Data

- Python integers – `int` class
  - Dynamic allocation implementation has significant storage overhead.

```
> print( type(100) )
    <class ‘int’>
> import sys
> sys.getsizeof(0)
```

How Computers Store Data

- Python integers – `int` class
  - Dynamic allocation implementation has significant storage overhead.

Number of bytes →

2^{30} = 1024x1024x1024 →

```
> print( type(100) )
    <class 'int'>
> import sys
> sys.getsizeof(0)
    28
> sys.getsizeof(1073741823)
    28
> sys.getsizeof(1073741824)
    32
```

How Computers Store Data

- Python integers – `int` class
  - No need to consider overflow (unless we're running out of physical memory)

```
> sys.getsizeof(1073741824)
    32
> sys.getsizeof(2**1000)
    160
> 2**1000
```

How Computers Store Data

- Python integers – `int` class
  - No need to consider overflow (unless we're running out of physical memory)

```
> sys.getsizeof(1073741824)
    32
> sys.getsizeof(2**1000)
    160
> 2**1000
    107150860718626732094842504906000181056140481170553
    360744375038837035105112493612249319837881569585812
    759467291755314682518714528569231404359845775746985
    748039345677748242309854210746050623711418779541821
    530464749835819412673987675591655439460770629145711
    964776865421676604298316526243868372056680693
```

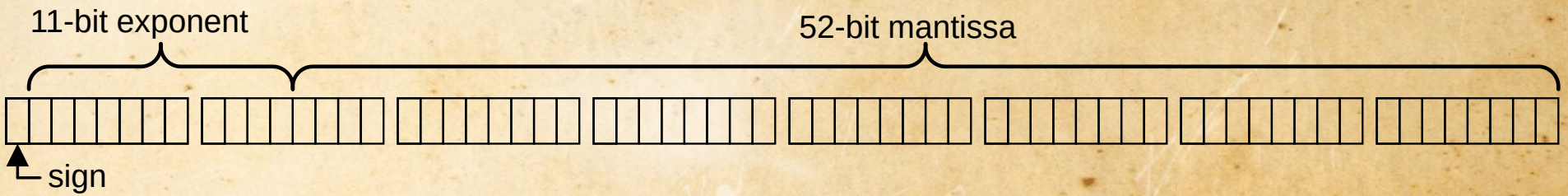
Lecture Contents

- How data is stored
  - Numbers
    - Integers
    - **Real Numbers**
  - Characters
  - Arrays

How Computers Store Data

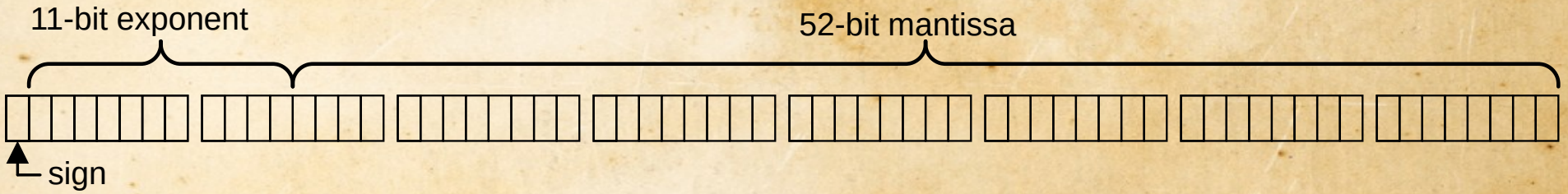
- Real Numbers
  - Modern processors follow the IEEE 754 standard
  - Similar to scientific notation

$$\pm\, mantissa \times 2^{exponent}$$



How Computers Store Data

- Real Numbers
  - Modern processors and most programming languages follow the IEEE 754 standard



	sign	exponent	mantissa
float	1	8	23
double	1	11	52

How Computers Store Data

- Python Real Numbers – `float` class
  - 64-bit floating point number (8 bytes)
    - Does NOT dynamically expand!
  - Object overhead (16 bytes)

```
> sys.getsizeof(0)
    28
> sys.getsizeof(0.0)
    24
> sys.getsizeof(2**1000)
    160
> sys.getsizeof(2.0**1000)
```

How Computers Store Data

- Python Real Numbers – `float` class
  - 64-bit floating point number (8 bytes)
    - Does NOT dynamically expand!
  - Object overhead (16 bytes)

```
> sys.getsizeof(0)
    28
> sys.getsizeof(0.0)
    24
> sys.getsizeof(2**1000)
    160
> sys.getsizeof(2.0**1000)
    24
> 2.0**1000
    1.071508607186267e+301
```

How Computers Store Data

- Python Real Numbers – `float` class
  - Note that not all Python interpreters use 64-bit floating point numbers
    - MicroPython uses 32-bits.
    - Some Python libraries use 32-bit or even 16-bit operations.

```
> sys.float_info.max
    1.7976931348623157e+308
> sys.float_info.min
    2.2250738585072014e-308
> sys.float_info.epsilon
    2.220446049250313e-16
```

How Computers Store Data

- Python Real Numbers – `float` class
  - Loss of precision not only occurs with irrational numbers, repeating decimals, or high-precision numbers

```
> 0.1 + 0.1 + 0.1 == 0.3
```

How Computers Store Data

- Python Real Numbers – `float` class
  - Loss of precision not only occurs with irrational numbers, repeating decimals, or high-precision numbers

```
> 0.1 + 0.1 + 0.1 == 0.3
False
> 0.1 + 0.1 + 0.1
```

How Computers Store Data

- Python Real Numbers – `float` class
  - Loss of precision not only occurs with irrational numbers, repeating decimals, or high-precision numbers

Why?!

```
> 0.1 + 0.1 + 0.1 == 0.3
False
> 0.1 + 0.1 + 0.1
    0.30000000000000004
```

How Computers Store Data

- Python Real Numbers – `float` class
  - Loss of precision not only occurs with irrational numbers, repeating decimals, or high-precision numbers

```
> 0.1 + 0.1 + 0.1 == 0.3
False
> 0.1 + 0.1 + 0.1
    0.30000000000000004
```

In binary...

$$(0.1)_{10} = \frac{0}{2} + \frac{0}{4} + \frac{0}{8} + \frac{1}{16} + \frac{1}{32} + \frac{0}{64} + \frac{0}{128} + \frac{1}{256} + \frac{1}{512} + \frac{0}{1024} + \ldots$$

$$= 0 + 0 + 0 + 0.0625 + 0.03125 + 0 + 0 + 0.00390625 + 0.00195312 + 0 + \ldots$$

$$(0.1)_{10} = (0.0001100110011001100110011001100110011001100110011\ldots)_2$$

List of Java Primitive Types

- *Integers*
  - **`byte`** (8 bits)
  - **`short`** (16 bits)
  - **`int`** (32 bits)
  - **`long`** (64 bits)
- *Real Numbers*
  - **`float`** (32 bits)
  - **`double`** (64 bits)
- *True/False*
  - **`boolean`** (1 bit?)
- *Letters*
  - **`char`** (16 bits)

Python Data Types

Numbers