CSCI 141
Lecture 5:
Code Execution
Order of Operations
Binary representation
Announcements
Announcements

• WWU has a Society of Women Engineers (SWE) club
Announcements

• WWU has a Society of Women Engineers (SWE) club

  • CS students are eligible to join. They have cool events and career networking opportunities, among other things
Announcements

• WWU has a Society of Women Engineers (SWE) club

  • CS students are eligible to join. They have cool events and career networking opportunities, among other things

  • Not just for women: men are also welcome to join
Announcements

• WWU has a Society of Women Engineers (SWE) club

  • CS students are eligible to join. They have cool events and career networking opportunities, among other things

  • Not just for women: men are also welcome to join

  • Their first meeting is at 6:00pm next Wednesday in ET 321
Announcements
Announcements

• A1 is due Monday!
Announcements

• A1 is due Monday!
  • Start soon if you haven't yet...
Announcements

• A1 is due Monday!
  • Start soon if you haven't yet...

• Lab 1 is due tonight!
Announcements

• A1 is due Monday!
  • Start soon if you haven't yet...

• Lab 1 is due tonight!
  • Make sure you've submitted your file on Canvas
QOTD

What will the following line print?

```python
print(int(str("43")))
```
QOTD

What will the following program print?

day = "12"
year = "Saturday"
print("mon", year, sep="day", end=day)
What will the following program print?

```python
a = 4 // 2
b = 3 // 2
c = 3 % 2
print(a + b + c)
```
Goals

• Understand how the + and * operators behave with string operands.

• Know how to apply operator precedence rules to determine the order in which pieces of an expression are evaluated.

• Know how to convert a decimal number to binary and vice versa.

• Understand the basic idea behind how strings and floating-point numbers are represented on computers.
Code execution: Putting it all together

```python
a = 4
b = float(2 + a)
```
Code execution: Putting it all together

• Consider this program:
  
a = 4
  
b = float(2 + a)

• What happens when we execute it?
Code execution: Putting it all together

- Consider this program:

  ```python
  a = 4
  b = float(2 + a)
  ```

- What happens when we execute it?
  - the value 4 gets stored in `a`
Code execution: Putting it all together

• Consider this program:
  ```
  a = 4
  b = float(2 + a)
  ```

• What happens when we execute it?
  • the value 4 gets stored in `a`
  • the expression `2+a` is evaluated, resulting in the value 6
Code execution: Putting it all together

• Consider this program:
  
  ```python
  a = 4
  b = float(6)
  ```

• What happens when we execute it?
  • the value 4 gets stored in `a`
  • the expression `2+a` is evaluated, resulting in the value 6
Code execution: Putting it all together

- Consider this program:
  
  ```
  a = 4
  b = float(6)
  ```

- What happens when we execute it?
  
  - the value 4 gets stored in `a`
  - the expression `2+a` is evaluated, resulting in the value 6
  - 6 is passed into the `float` function
Code execution: Putting it all together

• Consider this program:

```python
a = 4
b = 6.0
```

• What happens when we execute it?
  • the value 4 gets stored in `a`
  • the expression `2+a` is evaluated, resulting in the value 6
  • 6 is passed into the `float` function
  • the `float` function converts 6 to a `float` and returns `6.0`
Code execution: Putting it all together

• Consider this program:

```python
a = 4
b = 6.0
```

• What happens when we execute it?
  • the value 4 gets stored in `a`
  • the expression `2+a` is evaluated, resulting in the value 6
  • 6 is passed into the `float` function
  • the `float` function converts 6 to a `float` and returns `6.0`
  • the value `6.0` gets stored in variable `b`
In what order do things get evaluated?

A function’s arguments are always evaluated left-to-right before it is called:
Code execution: Putting it all together

In what order do things get evaluated?

A function’s arguments are always evaluated left-to-right before it is called:

```python
print(2+2, 4+6, int(10.4))
```
In what order do things get evaluated?

A function’s arguments are always evaluated left-to-right before it is called:

```python
print(2+2, 4+6, int(10.4))
print(4, 4+6, int(10.4))
```
Code execution: Putting it all together

In what order do things get evaluated?

A function’s arguments are always evaluated left-to-right before it is called:

```
print(2+2, 4+6, int(10.4))
print(4, 4+6, int(10.4))
print(4, 10, int(10.4))
```
Code execution: Putting it all together

In what order do things get evaluated?

A function’s arguments are always evaluated left-to-right before it is called:

```
print(2+2, 4+6, int(10.4))
print(4, 4+6, int(10.4))
print(4, 10, int(10.4))
print(4, 10, 10)
```
Code execution: Putting it all together

In what order do things get evaluated?

A function’s arguments are always evaluated left-to-right before it is called:

```
print(2+2, 4+6, int(10.4))
print(4, 4+6, int(10.4))
print(4, 10, int(10.4))
print(4, 10, 10)
```

4 10 10 is printed to the console
Code execution: Putting it all together

• In what order do things get evaluated?

• A function’s arguments are always evaluated left-to-right before it is called:

  \[ \text{print}(2+2, 4+6, \text{int}(10.4)) \]

• Parenthesized expressions are evaluated inside-out:
Code execution: Putting it all together

• In what order do things get evaluated?

• A function’s arguments are always evaluated left-to-right before it is called:
  
  ```python
  print(2+2, 4+6, int(10.4))
  ```

• Parenthesized expressions are evaluated inside-out:
  
  ```python
  20 // (6 + 3)
  ```
Code execution: Putting it all together

• In what order do things get evaluated?

• A function’s arguments are always evaluated left-to-right before it is called:

\[
\text{print}(2+2, 4+6, \text{int}(10.4))
\]

• Parenthesized expressions are evaluated inside-out:

\[
20 \ // \ (6 + 3) \\
20 \ // \ 9
\]
Code execution: Putting it all together

• In what order do things get evaluated?

• A function’s arguments are always evaluated left-to-right before it is called:

```python
print(2+2, 4+6, int(10.4))
```

```plaintext
20 // (6 + 3)
20 // 9
=> 2
```

• Parenthesizesed expressions are evaluated inside-out:
Code execution: Putting it all together

• In what order do things get evaluated?

• A function’s arguments are always evaluated left-to-right before it is called

```python
print(2+2, 4+6, int(10.4))
```

• Parenthesized expressions are evaluated inside-out:

```python
20 // (6 + 3)
```

• What about

```python
20 // 6 + 3
```
Code execution: Putting it all together

• In what order do things get evaluated?

• A function’s arguments are always evaluated left-to-right before it is called

  \[
  \text{print}(2+2, 4+6, \text{int}(10.4))
  \]

• Parenthesized expressions are evaluated inside-out:

  \[
  20 \div (6 + 3)
  \]

• What about \[
  20 \div 6 + 3
  \]

  More later on operator precedence.
A Note on Operators

• Operators only work if their operands have the correct types.

• Some operators can work on more than one type or combination of types:
A Note on Operators

• Operators only work if their operands have the correct types.

• Some operators can work on more than one type or combination of types:

Not too surprising:

int + int => int
int + float => float
float + int => float
float + float => float
A Note on Operators

• Operators only work if their operands have the correct types.

• Some operators can work on more than one type or combination of types:

Not too surprising:

int + int => int
int + float => float
float + int => float
float + float => float

Maybe a little surprising:

str + str => str
str * int => str
A Note on Operators

• Operators only work if their operands have the correct types.  \( \text{float} \times \text{str} \Rightarrow \text{error} \)

• Some operators can work on more than one type or combination of types:

Not too surprising:

\[
\begin{align*}
\text{int} + \text{int} & \Rightarrow \text{int} \\
\text{int} + \text{float} & \Rightarrow \text{float} \\
\text{float} + \text{int} & \Rightarrow \text{float} \\
\text{float} + \text{float} & \Rightarrow \text{float}
\end{align*}
\]

Maybe a little surprising:

\[
\begin{align*}
\text{str} + \text{str} & \Rightarrow \text{str} \\
\text{str} \times \text{int} & \Rightarrow \text{str}
\end{align*}
\]
Demo
Demo

• operator behaviors:

  4 + 5 => 9
  4.0 + 5 => 9.0
  4.0 + 5.0 => 9.0
  "a" + "b" => "ab"
  "a" + 1 => error
  "a" + "b" => "ab"
  "a" * 16 => "aaaaaaaaaaaaaaaaaaa"
Suppose we run the following program, and the user types 6 and presses enter.

What value gets stored in result?

```python
user_num = input("Enter a number: ")
result = 5 % (3 ** (user_num // 4))
```
Practice Problem: Operators

Suppose we run the following program, and the user types 6 and presses enter.

What value gets stored in result?

```python
user_num = input("Enter a number: ")
result = 5 % (3 ** (user_num // 4))
```

A: 1
B: 2
C: 3
D: None of the above
Practice Problem: Operators

Suppose we run the following program, and the user types 6 and presses enter.

What value gets stored in result?

```
user_num = input("Enter a number: ")
result = 5 % (3 ** (user_num // 4))
```

Let’s try it out...
Practice Problem: Operators

Suppose we run the following program, and the user types 6 and presses enter.

What value gets stored in \texttt{result}?

```
user_num = \texttt{input}("Enter a number: ")
result = 5 \% (3 ** (user_num // 4))
```

A: 1
B: 2
C: 3
D: None of the above
Bugs

• We had a bug in our code!

• Why are they called bugs? An anecdote from the history of computing:

At 3:45 p.m., Grace Murray Hopper records 'the first computer bug' in the Harvard Mark II computer's log book. The problem was traced to a moth stuck between relay contacts in the computer, which Hopper duly taped into the Mark II's log book with the explanation: “First actual case of bug being found.” The bug was actually found by others but Hopper made the logbook entry.

Source: https://www.computerhistory.org/tdih/september/9/
“First actual case of a bug being found”
Practice Problem: Operators

Suppose we run the following program, and the user types 6 and presses enter.

What value gets stored in result?

```python
user_num = int(input("Enter a number: "))
result = 5 % (3 ** (user_num // 4))
```
Suppose we run the following program, and the user types 6 and presses enter.

What value gets stored in result?

```python
user_num = int(input("Enter a number: "))
result = 5 % (3 ** (user_num // 4))
```
Practice Problem: Operators

Suppose we run the following program, and the user types 6 and presses enter.

What value gets stored in `result`?

```python
user_num = int(input("Enter a number: "))
result = 5 % (3 ** (user_num // 4))
```

A: 1
B: 2
C: 3
D: None of the above
Practice Problem: Operators

Suppose we run the following program, and the user types 6 and presses enter.

What value gets stored in result?

```python
user_num = int(input("Enter a number: "))
result = 5 % (3 ** (user_num // 4))
```
Suppose we run the following program, and the user types 6 and presses enter.

What value gets stored in result?

```python
user_num = int(input("Enter a number: "))
result = 5 % (3 ** (user_num // 4))
```
Practice Problem: Operators

```python
user_num = int(input("Enter a number: "))
result = 5 % (3 ** (user_num // 4))
result = 5 % (3 ** (6 // 4))
```
Practice Problem: Operators

```python
user_num = int(input("Enter a number: "))
result = 5 % (3 ** (user_num // 4))
result = 5 % (3 ** (6 // 4))
```
Practice Problem: Operators

user_num = \textbf{int}(\textbf{input}("Enter a number: "))
result = 5 \% (3 ** (user_num // 4))
result = 5 \% (3 ** (6 // 4))
result = 5 \% (3 ** 1)
Practice Problem: Operators

```python
user_num = int(input("Enter a number: "))
result = 5 % (3 ** (user_num // 4))
result = 5 % (3 ** (6 // 4))
result = 5 % (3 ** 1)
```
Practice Problem: Operators

```python
user_num = int(input("Enter a number: "))
result = 5 % (3 ** (user_num // 4))
result = 5 % (3 ** (6 // 4))
result = 5 % (3 ** 1)
result = 5 % (3)
```
Practice Problem: Operators

```python
user_num = int(input("Enter a number: "))
result = 5 % (3 ** (user_num // 4))
result = 5 % (3 ** (6 // 4))
result = 5 % (3 ** 1)
result = 5 % (3 )
```
Practice Problem: Operators

user_num = int(input("Enter a number: "))
result = 5 % (3 ** (user_num // 4))
result = 5 % (3 ** (6 // 4))
result = 5 % (3 ** 1)
result = 5 % (3)
result = 2
Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules?

What if we took the parentheses out?

\[
\text{result} = 5 \% (3 ** (6 \div 4))
\]

\[
\text{result} = 5 \% 3 ** 6 \div 4
\]
Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: operator precedence.
Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: operator precedence.

Remember PEMDAS? BIDMAS? BODMAS?
Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: operator precedence.

Remember PEMDAS? BIDMAS? BODMAS?

- Parentheses
- Exponentiation
- Multiplication and Division
- Addition and Subtraction
Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: operator precedence.

Remember PEMDAS? BIDMAS? BODMAS?

- Parentheses
- Exponentiation
- Multiplication and Division
- Addition and Subtraction
Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: **operator precedence**.

Remember PEMDAS? BIDMAS? BODMAS?

- Parentheses
- Exponentiation
- Multiplication and Division
- Addition and Subtraction

precedence

order of evaluation
Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: **operator precedence**.

Remember PEMDAS? BIDMAS? BODMAS?

- Parentheses
- Exponentiation
- Multiplication and Division (left-to-right)
- Addition and Subtraction (left-to-right)
Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: **operator precedence**.

Remember PEMDAS? BIDMAS? BODMAS?

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Order of Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parentheses</td>
<td></td>
</tr>
<tr>
<td>Exponentiation</td>
<td></td>
</tr>
<tr>
<td>Multiplication and Division (left-to-right)</td>
<td></td>
</tr>
<tr>
<td>Addition and Subtraction (left-to-right)</td>
<td></td>
</tr>
</tbody>
</table>

Example (whiteboard): $10 \times 6 ** 2 / 5 \div 11$
Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: **operator precedence**.

Remember PEMDAS? BIDMAS? BODMAS?

Example: $2 ** 2 ** 3$

- Parentheses
- Exponentiation
- Multiplication and Division (left-to-right)
- Addition and Subtraction (left-to-right)

Example: $2 ** 2 ** 3$
We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: **operator precedence**.

Remember PEMDAS? BIDMAS? BODMAS?

**Example:**

\[ 2 \ ** \ 2 \ ** \ 3 \]
Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: operator precedence.

Remember PEMDAS? BIDMAS? BODMAS?

Example:

2 ** 2 ** 3

precedence

Parentheses
Exponentiation
Multiplication and Division (left-to-right)
Addition and Subtraction (left-to-right)

order of evaluation

Example:

2 ** 2 ** 3
Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: operator precedence.

Remember PEMDAS? BIDMAS? BODMAS?

Example:

\[
2 \times 2 \times 3
\]

(2 ** 2) ** 3
Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: operator precedence.

Remember PEMDAS? BIDMAS? BODMAS?

Example:

\[
\begin{align*}
2 & \quad ** \quad 2 \quad ** \quad 3 \\
(2 & \quad ** \quad 2) \quad ** \quad 3 \\
& \quad \Rightarrow \quad 4^3 \quad \Rightarrow \quad 64
\end{align*}
\]
Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: operator precedence.

Remember PEMDAS? BIDMAS? BODMAS?

- Parentheses
- Exponentiation
- Multiplication and Division (left-to-right)
- Addition and Subtraction (left-to-right)

Example:

\[
2 \times 2^2^3
\]

\[
2 \times (2^2^3)
\]
Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: **operator precedence**.

Remember PEMDAS? BIDMAS? BODMAS?

- Parentheses
- Exponentiation
- Multiplication and Division (left-to-right)
- Addition and Subtraction (left-to-right)

Example:

\[
\begin{align*}
2 \; ** \; 2 \; ** \; 3 &= (2 \; ** \; 2) \; ** \; 3 \\
&= 4^3 \\
&= 64 \\
2 \; ** \; (2 \; ** \; 3) &= 2^8 \\
&= 256
\end{align*}
\]
Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: operator precedence.

Remember PEMDAS? BIDMAS? BODMAS?

- **Parentheses**
- **Exponentiation**
- **Multiplication and Division** (left-to-right)
- **Addition and Subtraction** (left-to-right)

**Example:**

\[
2^{2^{3}}
\]

1. Parentheses: \((2^2)^3\)
2. Exponentiation: \(4^3\)
3. Order of evaluation: \(64\)

\[
2^{(2^3)}
\]

1. Parentheses: \(2^{8}\)
2. Exponentiation: \(256\)

**Order of Evaluation:**

1. Parentheses
2. Exponentiation
3. Multiplication and Division (left-to-right)
4. Addition and Subtraction (left-to-right)
Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: **operator precedence**.

Remember PEMDAS? BIDMAS? BODMAS?

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Order of Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parentheses</td>
<td></td>
</tr>
<tr>
<td>Exponentiation</td>
<td><strong>right-to-left</strong></td>
</tr>
<tr>
<td>Multiplication and Division</td>
<td><strong>left-to-right</strong></td>
</tr>
<tr>
<td>Addition and Subtraction</td>
<td><strong>left-to-right</strong></td>
</tr>
</tbody>
</table>

Example:

\[
2 \times 2 \times 3 \\
(2 \times 2) \times 3 \\
2 \times (2 \times 3) \\
\]

=> 6 
=> 64 
=> 256
What does the following expression evaluate to?

1 + 2 ** 3 / 4 * 5 - (6 % 7)

A. 4
B. 5
C. 6
D. 4.0
E. 5.0
F. 6.0
Questions?
Representing Numbers on Computers

• What happens “under the hood” when we execute:

  \[\text{result} = 5\]

• The value 5 gets stored somewhere in main memory (and we somehow keep track of where it’s stored).
Representing Numbers on Computers

- What happens “under the hood” when we execute:
  \[ \text{result} = 5 \]

- The value 5 gets stored somewhere in main memory (and we somehow keep track of where it’s stored).
Representing Numbers on Computers

How are numbers stored in memory?

Memory is made of specialized electric circuits that provide cells that can “store” information by being in one of two states: on or off.
Representing Numbers on Computers

How are numbers stored in memory?

Memory is made of specialized electric circuits that provide cells that can “store” information by being in one of two states: on or off.
Representing Numbers on Computers

How are numbers stored in memory?

We impose mathematical meaning on these states:
“off” = 0
“on” = 1
Representing Numbers on Computers

How are numbers stored in memory?

We impose mathematical meaning on these states:
“off” = 0
“on” = 1
Representing Numbers on Computers

How are numbers stored in memory?

We impose mathematical meaning on these states:
“off” = 0
“on” = 1
Representing Numbers on Computers

How are numbers stored in memory?

Each 1/0 memory location is called a bit.
Representing Numbers on Computers

Each 0/1 memory location stores one bit.
8 bits is called a byte.

Metric prefixes are used to represent numbers of bytes, e.g. kilo, mega, giga, etc.

In computer science, kilo is not actually 1000, it’s 1024.
Representing Numbers on Computers

Each 0/1 memory location stores one bit.
8 bits is called a byte.

Metric prefixes are used to represent numbers of bytes, e.g. kilo, mega, giga, etc.

In computer science, the prefixes have slightly different meaning: kilo is not actually 1000, it’s 1024.
Representing Numbers on Computers

Each 0/1 memory location stores one bit.
8 bits is called a byte.

Usual SI prefixes:
- kilo = $10^3 = 1000$
- mega = $10^6 = 1$ million
- giga = $10^9 = 1$ billion
- tera = $10^{12} = 1$ trillion
Representing Numbers on Computers

Each 0/1 memory location stores one bit. 8 bits is called a byte.

Usual SI prefixes:
- kilo = $10^3 = 1000$
- mega = $10^6 = 1$ million
- giga = $10^9 = 1$ billion
- tera = $10^{12} = 1$ trillion

Base 2 prefixes:
- kilobyte = $2^{10} = 1,024$ bytes
- megabyte = $2^{20} = 1,048,576$ bytes
- gigabyte = $2^{30} = 1,073,741,824$ bytes
- terabyte = $2^{40} = 1,099,511,627,776$ bytes
Binary Representation

In decimal:
\[
104 = 1 \times 10^2 \quad \text{(hundreds place)}
\]
\[
+ \quad 0 \times 10^1 \quad \text{(tens place)}
\]
\[
+ \quad 4 \times 10^0 \quad \text{(ones place)}
\]
Binary Representation

If all we can store is 0’s and 1’s, how do we represent other numbers (e.g., 23?)

• By representing numbers in base 2 (binary) instead of base 10 (decimal).

  In decimal:

  • Observation:  \[ 104 = 1 \times 10^2 \quad \text{(hundreds place)} \]
  \[ + \quad 0 \times 10^1 \quad \text{(tens place)} \]
  \[ + \quad 4 \times 10^0 \quad \text{(ones place)} \]
Binary Representation

If all we can store is 0’s and 1’s, how do we represent other numbers (e.g., 23?)

• By representing numbers in base 2 (binary) instead of base 10 (decimal).

In decimal:

• Observation: \[
\begin{align*}
104 &= 1 \times 10^2 \quad \text{(hundreds place)} \\
&\quad + 0 \times 10^1 \quad \text{(tens place)} \\
&\quad + 4 \times 10^0 \quad \text{(ones place)}
\end{align*}
\]
If all we can store is 0’s and 1’s, how do we represent other numbers (e.g., 23?)

- By representing numbers in \textit{base 2} (binary) instead of \textit{base 10} (decimal).
  
  \begin{align*}
  \text{In decimal:} \\
  104 &= 1 \times 10^2 \quad \text{(hundreds place)} \\
  &\quad + 0 \times 10^1 \quad \text{(tens place)} \\
  &\quad + 4 \times 10^0 \quad \text{(ones place)}
  \end{align*}

- The decimal representation of a number is a sum of multiples of the powers of ten.
If all we can store is 0’s and 1’s, how do we represent other numbers (e.g., 23?)

• By representing numbers in base 2 (binary) instead of base 10 (decimal).

In decimal:
• Observation:  $104 = 1 \times 10^2$ (hundreds place) 
  $+ 0 \times 10^1$ (tens place) 
  $+ 4 \times 10^0$ (ones place)

• Key idea: use 2 here instead of 10.
Binary to Decimal

- In decimal, each digit represents a multiple of a power of 2
Binary to Decimal

- In decimal, each digit represents a multiple of a power of 2.
## Binary to Decimal

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2^5)</td>
<td>(2^4)</td>
<td>(2^3)</td>
<td>(2^2)</td>
<td>(2^1)</td>
<td>(2^0)</td>
<td></td>
</tr>
</tbody>
</table>
Binary to Decimal

In decimal, each digit represents a multiple of a power of 2.
Binary to Decimal

- In decimal, each digit represents a multiple of a power of 2.
Binary to Decimal

- In decimal, each digit represents a multiple of a power of 2
Binary to Decimal

- In decimal, each digit represents a multiple of a power of 2
Binary to Decimal

- In decimal, each digit represents a multiple of a power of 2

```
1 0 1 1 1 1
2^5 2^4 2^3 2^2 2^1 2^0
4 2 1
```
Binary to Decimal

- In decimal, each digit represents a multiple of a power of 2.
Binary to Decimal

- In decimal, each digit represents a multiple of a power of 2
Binary to Decimal

In decimal, each digit represents a multiple of a power of 2.

\[
\begin{align*}
1 
& \times 2^5 \\
0 
& \times 2^4 \\
1 
& \times 2^3 \\
1 
& \times 2^2 \\
1 
& \times 2^1 \\
1 
& \times 2^0 \\
\end{align*}
\]

32 + 8 + 4 + 2 + 1
Binary to Decimal

- In decimal, each digit represents a multiple of a power of 2

```
  2^5  2^4  2^3  2^2  2^1  2^0
  32   8    4    2    1

= 47
```
Binary to Decimal

- In decimal, each digit represents a multiple of a power of 2
- 10111 in binary is 47 in decimal.
Decimal to Binary

Converting decimal to binary goes the other way. Problem: write 23 as a sum of powers of 2
Decimal to Binary

Converting decimal to binary goes the other way. Problem: write 23 as a sum of powers of 2

23 = \(? * 2^4\) (16)  
+ \(? * 2^3\) (8)  
+ \(? * 2^2\) (4)  
+ \(? * 2^1\) (2)  
+ \(? * 2^0\) (1)
Decimal to Binary

Converting decimal to binary goes the other way. Problem: write 23 as a sum of powers of 2

\[
23 = \ ? \times 2^4 (16) \\
+ \ ? \times 2^3 (8) \\
+ \ ? \times 2^2 (4) \\
+ \ ? \times 2^1 (2) \\
+ \ ? \times 2^0 (1)
\]

The binary representation of the decimal number 23 is:

A. 10111
B. 11101
C. 01100
D. 11110
Decimal to Binary

Converting decimal to binary goes the other way. Problem: write 23 as a sum of powers of 2

\[
23 = ? \times 2^4 \quad (16)
+ \quad ? \times 2^3 \quad (8)
+ \quad ? \times 2^2 \quad (4)
+ \quad ? \times 2^1 \quad (2)
+ \quad ? \times 2^0 \quad (1)
\]

The binary representation of the decimal number 23 is:

A. 10111
B. 11101
C. 01100
D. 11110
Decimal to Binary

Converting decimal to binary goes the other way.
Problem: write 23 as a sum of powers of 2

\[
23 = \ ? \times 2^4 \ (16) \ + \ ? \times 2^3 \ (8) \ + \ ? \times 2^2 \ (4) \ + \ ? \times 2^1 \ (2) \ + \ ? \times 2^0 \ (1)
\]

(23-16 = 7 left)

The binary representation of the decimal number 23 is:

A. 10111
B. 11101
C. 01100
D. 11110
Decimal to Binary

Converting decimal to binary goes the other way. Problem: write 23 as a sum of powers of 2

\[
23 = ? \times 2^4 (16) + ? \times 2^3 (8) + ? \times 2^2 (4) + ? \times 2^1 (2) + ? \times 2^0 (1)
\]

The binary representation of the decimal number 23 is:

A. 10111
B. 11101
C. 01100
D. 11110
Decimal to Binary

Converting decimal to binary goes the other way.
Problem: write 23 as a sum of powers of 2

\[
23 = ? \times 2^4 \quad (16) \quad 1 \quad (23-16 = 7 \text{ left})
+ \quad ? \times 2^3 \quad (8) \quad 0 \quad (7-0 = 7 \text{ left})
+ \quad ? \times 2^2 \quad (4) \quad 1 \quad (7-4 = 3 \text{ left})
+ \quad ? \times 2^1 \quad (2)
+ \quad ? \times 2^0 \quad (1)
\]

The binary representation of the decimal number 23 is:

A. 10111
B. 11101
C. 01100
D. 11110
Decimal to Binary

Converting decimal to binary goes the other way. Problem: write 23 as a sum of powers of 2

\[
23 = ? \times 2^4 (16) + ? \times 2^3 (8) + ? \times 2^2 (4) + ? \times 2^1 (2) + ? \times 2^0 (1)
\]

\[
= ? \times 2^4 (16) + 1 \quad (23-16 = 7 \text{ left})
\]
\[
+ ? \times 2^3 (8) + 0 \quad (7-0 = 7 \text{ left})
\]
\[
+ ? \times 2^2 (4) + 1 \quad (7-4 = 3 \text{ left})
\]
\[
+ ? \times 2^1 (2) + 1 \quad (3-2 = 1 \text{ left})
\]
\[
+ ? \times 2^0 (1)
\]

The binary representation of the decimal number 23 is:

A. 10111
B. 11101
C. 01100
D. 11110
Decimal to Binary

Converting decimal to binary goes the other way. Problem: write 23 as a sum of powers of 2

\[
23 = ? \times 2^4 \quad (16) \quad 1 \quad (23-16 = 7 \text{ left}) \\
+ \quad ? \times 2^3 \quad (8) \quad 0 \quad (7-0 = 7 \text{ left}) \\
+ \quad ? \times 2^2 \quad (4) \quad 1 \quad (7-4 = 3 \text{ left}) \\
+ \quad ? \times 2^1 \quad (2) \quad 1 \quad (3-2 = 1 \text{ left}) \\
+ \quad ? \times 2^0 \quad (1) \quad 1 \quad (1-1 = 0 \text{ left})
\]

The binary representation of the decimal number 23 is:

A. 10111
B. 11101
C. 01100
D. 11110
Decimal to Binary

Converting decimal to binary goes the other way.
Problem: write 23 as a sum of powers of 2

\[
23 = ? \times 2^4 (16) + ? \times 2^3 (8) + ? \times 2^2 (4) + ? \times 2^1 (2) + ? \times 2^0 (1)
\]

\[
= 1 \times 2^4 (16) + 0 \times 2^3 (8) + 1 \times 2^2 (4) + 1 \times 2^1 (2) + 1 \times 2^0 (1)
\]

(23-16 = 7 left)
(7-0 = 7 left)
(7-4 = 3 left)
(3-2 = 1 left)
(1-1 = 0 left)

The binary representation of the decimal number 23 is:

A. 10111
B. 11101
C. 01100
D. 11110
That’s how int works.

- What about str and float?
A string is a sequence of letters (or characters).

1. Agree by convention on a number that represents each character.
2. Convert that number to binary.
3. Store a sequence of those numbers to form a string.
How do you store strings?

Various conventions exist:
ASCII, Unicode

A str is a sequence of letters (or characters).

1. Agree by convention on a number that represents each character.
2. Convert that number to binary.
3. Store a sequence of those numbers to form a string.
How do you store strings?

**ASCII TABLE**

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>[NULL]</td>
<td>32</td>
<td>20</td>
<td>[SPACE]</td>
<td>64</td>
<td>40</td>
<td>@</td>
<td>96</td>
<td>60</td>
<td>`</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>[START OF HEADING]</td>
<td>33</td>
<td>21</td>
<td>!</td>
<td>65</td>
<td>41</td>
<td>A</td>
<td>97</td>
<td>61</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>[START OF TEXT]</td>
<td>34</td>
<td>22</td>
<td>&quot;</td>
<td>66</td>
<td>42</td>
<td>B</td>
<td>98</td>
<td>62</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>[END OF TEXT]</td>
<td>35</td>
<td>23</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>C</td>
<td>99</td>
<td>63</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>[END OF TRANSMISSION]</td>
<td>36</td>
<td>24</td>
<td>$</td>
<td>68</td>
<td>44</td>
<td>D</td>
<td>100</td>
<td>64</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>[ENQUIRY]</td>
<td>37</td>
<td>25</td>
<td>%</td>
<td>69</td>
<td>45</td>
<td>E</td>
<td>101</td>
<td>65</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>[ACKNOWLEDGE]</td>
<td>38</td>
<td>26</td>
<td>&amp;</td>
<td>70</td>
<td>46</td>
<td>F</td>
<td>102</td>
<td>66</td>
<td>f</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>[BEL]</td>
<td>39</td>
<td>27</td>
<td>'</td>
<td>71</td>
<td>47</td>
<td>G</td>
<td>103</td>
<td>67</td>
<td>g</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>[BACKSPACE]</td>
<td>40</td>
<td>28</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>H</td>
<td>104</td>
<td>68</td>
<td>h</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>[HORIZONTAL TAB]</td>
<td>41</td>
<td>29</td>
<td>)</td>
<td>73</td>
<td>49</td>
<td>I</td>
<td>105</td>
<td>69</td>
<td>i</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>[VERTICAL TAB]</td>
<td>43</td>
<td>2B</td>
<td>+</td>
<td>75</td>
<td>4B</td>
<td>K</td>
<td>107</td>
<td>6B</td>
<td>k</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>[FORM FEED]</td>
<td>44</td>
<td>2C</td>
<td>,</td>
<td>76</td>
<td>4C</td>
<td>L</td>
<td>108</td>
<td>6C</td>
<td>l</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>[CARRIAGE RETURN]</td>
<td>45</td>
<td>2D</td>
<td>-</td>
<td>77</td>
<td>4D</td>
<td>M</td>
<td>109</td>
<td>6D</td>
<td>m</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>[SHIFT OUT]</td>
<td>46</td>
<td>2E</td>
<td>.</td>
<td>78</td>
<td>4E</td>
<td>N</td>
<td>110</td>
<td>6E</td>
<td>n</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>[SHIFT IN]</td>
<td>47</td>
<td>2F</td>
<td>/</td>
<td>79</td>
<td>4F</td>
<td>O</td>
<td>111</td>
<td>6F</td>
<td>o</td>
</tr>
<tr>
<td>16</td>
<td>G</td>
<td>[DATA LINK ESCAPE]</td>
<td>48</td>
<td>30</td>
<td>0</td>
<td>80</td>
<td>50</td>
<td>P</td>
<td>112</td>
<td>70</td>
<td>p</td>
</tr>
<tr>
<td>17</td>
<td>H</td>
<td>[DEVICE CONTROL 1]</td>
<td>49</td>
<td>31</td>
<td>1</td>
<td>81</td>
<td>51</td>
<td>Q</td>
<td>113</td>
<td>71</td>
<td>q</td>
</tr>
<tr>
<td>18</td>
<td>I</td>
<td>[DEVICE CONTROL 2]</td>
<td>50</td>
<td>32</td>
<td>2</td>
<td>82</td>
<td>52</td>
<td>R</td>
<td>114</td>
<td>72</td>
<td>r</td>
</tr>
<tr>
<td>19</td>
<td>J</td>
<td>[DEVICE CONTROL 3]</td>
<td>51</td>
<td>33</td>
<td>3</td>
<td>83</td>
<td>53</td>
<td>S</td>
<td>115</td>
<td>73</td>
<td>s</td>
</tr>
<tr>
<td>20</td>
<td>K</td>
<td>[DEVICE CONTROL 4]</td>
<td>52</td>
<td>34</td>
<td>4</td>
<td>84</td>
<td>54</td>
<td>T</td>
<td>116</td>
<td>74</td>
<td>t</td>
</tr>
<tr>
<td>21</td>
<td>L</td>
<td>[NEGATIVE ACKNOWLEDGE]</td>
<td>53</td>
<td>35</td>
<td>5</td>
<td>85</td>
<td>55</td>
<td>U</td>
<td>117</td>
<td>75</td>
<td>u</td>
</tr>
<tr>
<td>22</td>
<td>M</td>
<td>[SYNCHRONOUS IDLE]</td>
<td>54</td>
<td>36</td>
<td>6</td>
<td>86</td>
<td>56</td>
<td>V</td>
<td>118</td>
<td>76</td>
<td>v</td>
</tr>
<tr>
<td>23</td>
<td>N</td>
<td>[ENG. OF TRANS. BLOCK]</td>
<td>55</td>
<td>37</td>
<td>7</td>
<td>87</td>
<td>57</td>
<td>W</td>
<td>119</td>
<td>77</td>
<td>w</td>
</tr>
<tr>
<td>24</td>
<td>O</td>
<td>[CANCEL]</td>
<td>56</td>
<td>38</td>
<td>8</td>
<td>88</td>
<td>58</td>
<td>X</td>
<td>120</td>
<td>78</td>
<td>x</td>
</tr>
<tr>
<td>25</td>
<td>P</td>
<td>[END OF MEDIUM]</td>
<td>57</td>
<td>39</td>
<td>9</td>
<td>89</td>
<td>59</td>
<td>Y</td>
<td>121</td>
<td>79</td>
<td>y</td>
</tr>
<tr>
<td>26</td>
<td>Q</td>
<td>[SUBSTITUTE]</td>
<td>58</td>
<td>3A</td>
<td>:</td>
<td>90</td>
<td>5A</td>
<td>Z</td>
<td>122</td>
<td>7A</td>
<td>z</td>
</tr>
<tr>
<td>27</td>
<td>R</td>
<td>[ESCAPE]</td>
<td>59</td>
<td>3B</td>
<td>;</td>
<td>91</td>
<td>5B</td>
<td>[</td>
<td>123</td>
<td>7B</td>
<td>{</td>
</tr>
<tr>
<td>28</td>
<td>S</td>
<td>[FILE SEPARATOR]</td>
<td>60</td>
<td>3C</td>
<td>&lt;</td>
<td>92</td>
<td>5C</td>
<td>\</td>
<td>124</td>
<td>7C</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>T</td>
<td>[GROUP SEPARATOR]</td>
<td>61</td>
<td>3D</td>
<td>=</td>
<td>93</td>
<td>5D</td>
<td>]</td>
<td>125</td>
<td>7D</td>
<td>}</td>
</tr>
<tr>
<td>30</td>
<td>U</td>
<td>[RECORD SEPARATOR]</td>
<td>62</td>
<td>3E</td>
<td>&gt;</td>
<td>94</td>
<td>5E</td>
<td>^</td>
<td>126</td>
<td>7E</td>
<td>~</td>
</tr>
<tr>
<td>31</td>
<td>V</td>
<td>[UNIT SEPARATOR]</td>
<td>63</td>
<td>3F</td>
<td>?</td>
<td>95</td>
<td>5F</td>
<td>_</td>
<td>127</td>
<td>7F</td>
<td>[DEL]</td>
</tr>
<tr>
<td>Decimal</td>
<td>Hex</td>
<td>Char</td>
<td>Decimal</td>
<td>Hex</td>
<td>Char</td>
<td>Decimal</td>
<td>Hex</td>
<td>Char</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-----</td>
<td>--------</td>
<td>---------</td>
<td>-----</td>
<td>--------</td>
<td>---------</td>
<td>-----</td>
<td>--------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>20</td>
<td>SPACE</td>
<td>64</td>
<td>40</td>
<td>@</td>
<td>96</td>
<td>60</td>
<td>`</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>21</td>
<td>!</td>
<td>65</td>
<td>41</td>
<td>A</td>
<td>97</td>
<td>61</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>22</td>
<td>&quot;</td>
<td>66</td>
<td>42</td>
<td>B</td>
<td>98</td>
<td>62</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>23</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>C</td>
<td>99</td>
<td>63</td>
<td>c</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>24</td>
<td>$</td>
<td>68</td>
<td>44</td>
<td>D</td>
<td>100</td>
<td>64</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>25</td>
<td>%</td>
<td>69</td>
<td>45</td>
<td>E</td>
<td>101</td>
<td>65</td>
<td>e</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>26</td>
<td>&amp;</td>
<td>70</td>
<td>46</td>
<td>F</td>
<td>102</td>
<td>66</td>
<td>f</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>27</td>
<td>'</td>
<td>71</td>
<td>47</td>
<td>G</td>
<td>103</td>
<td>67</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>28</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>H</td>
<td>104</td>
<td>68</td>
<td>h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>29</td>
<td>)</td>
<td>73</td>
<td>49</td>
<td>I</td>
<td>105</td>
<td>69</td>
<td>i</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>2A</td>
<td>*</td>
<td>74</td>
<td>4A</td>
<td>J</td>
<td>106</td>
<td>6A</td>
<td>j</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>2B</td>
<td>+</td>
<td>75</td>
<td>4B</td>
<td>K</td>
<td>107</td>
<td>6B</td>
<td>k</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>2C</td>
<td>,</td>
<td>76</td>
<td>4C</td>
<td>L</td>
<td>108</td>
<td>6C</td>
<td>l</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>2D</td>
<td>-</td>
<td>77</td>
<td>4D</td>
<td>M</td>
<td>109</td>
<td>6D</td>
<td>m</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>2E</td>
<td>.</td>
<td>78</td>
<td>4E</td>
<td>N</td>
<td>110</td>
<td>6E</td>
<td>n</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>2F</td>
<td>/</td>
<td>79</td>
<td>4F</td>
<td>O</td>
<td>111</td>
<td>6F</td>
<td>o</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>30</td>
<td>0</td>
<td>80</td>
<td>50</td>
<td>P</td>
<td>112</td>
<td>70</td>
<td>p</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>31</td>
<td>1</td>
<td>81</td>
<td>51</td>
<td>Q</td>
<td>113</td>
<td>71</td>
<td>q</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>32</td>
<td>2</td>
<td>82</td>
<td>52</td>
<td>R</td>
<td>114</td>
<td>72</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>33</td>
<td>3</td>
<td>83</td>
<td>53</td>
<td>S</td>
<td>115</td>
<td>73</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>34</td>
<td>4</td>
<td>84</td>
<td>54</td>
<td>T</td>
<td>116</td>
<td>74</td>
<td>t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>35</td>
<td>5</td>
<td>85</td>
<td>55</td>
<td>U</td>
<td>117</td>
<td>75</td>
<td>u</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>36</td>
<td>6</td>
<td>86</td>
<td>56</td>
<td>V</td>
<td>118</td>
<td>76</td>
<td>v</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>37</td>
<td>7</td>
<td>87</td>
<td>57</td>
<td>W</td>
<td>119</td>
<td>77</td>
<td>w</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>38</td>
<td>8</td>
<td>88</td>
<td>58</td>
<td>X</td>
<td>120</td>
<td>78</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>39</td>
<td>9</td>
<td>89</td>
<td>59</td>
<td>Y</td>
<td>121</td>
<td>79</td>
<td>y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>3A</td>
<td>:</td>
<td>90</td>
<td>5A</td>
<td>Z</td>
<td>122</td>
<td>7A</td>
<td>z</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>3B</td>
<td>;</td>
<td>91</td>
<td>5B</td>
<td>[</td>
<td>123</td>
<td>7B</td>
<td>{</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>3C</td>
<td>&lt;</td>
<td>92</td>
<td>5C</td>
<td>\</td>
<td>124</td>
<td>7C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>3D</td>
<td>=</td>
<td>93</td>
<td>5D</td>
<td>]</td>
<td>125</td>
<td>7D</td>
<td>}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### ASCII Table

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>[NULL]</td>
<td>32</td>
<td>20</td>
<td>[SPACE]</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>[START OF HEADING]</td>
<td>33</td>
<td>21</td>
<td>!</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>[START OF TEXT]</td>
<td>34</td>
<td>22</td>
<td>&quot;</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>[END OF TEXT]</td>
<td>35</td>
<td>23</td>
<td>#</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>[END OF TRANSMISSION]</td>
<td>36</td>
<td>24</td>
<td>$</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>[ENQUIRY]</td>
<td>37</td>
<td>25</td>
<td>%</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>[ACKNOWLEDGE]</td>
<td>38</td>
<td>26</td>
<td>&amp;</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>[BELL]</td>
<td>39</td>
<td>27</td>
<td>^</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>[BACKSPACE]</td>
<td>40</td>
<td>28</td>
<td>(</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>[HORIZONTAL TAB]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>[LINE FEED]</td>
<td>41</td>
<td>29</td>
<td>)</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>[VERTICAL TAB]</td>
<td>42</td>
<td>2A</td>
<td>{</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>[FORM FEED]</td>
<td>43</td>
<td>2B</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>[CARRIAGE RETURN]</td>
<td>44</td>
<td>2C</td>
<td>,</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>[SHIFT OUT]</td>
<td>45</td>
<td>2D</td>
<td>.</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>[SHIFT IN]</td>
<td>46</td>
<td>2E</td>
<td>:</td>
</tr>
<tr>
<td>16</td>
<td>G</td>
<td>[DATA LINK ESCAPE]</td>
<td>47</td>
<td>2F</td>
<td>/</td>
</tr>
<tr>
<td>17</td>
<td>H</td>
<td>[DEVICE CONTROL 1]</td>
<td>48</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>J</td>
<td>[DEVICE CONTROL 2]</td>
<td>49</td>
<td>31</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>K</td>
<td>[DEVICE CONTROL 3]</td>
<td>50</td>
<td>32</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>L</td>
<td>[DEVICE CONTROL 4]</td>
<td>51</td>
<td>33</td>
<td>3</td>
</tr>
<tr>
<td>21</td>
<td>M</td>
<td>[NEGATIVE ACKNOWLEDGE]</td>
<td>52</td>
<td>34</td>
<td>4</td>
</tr>
<tr>
<td>22</td>
<td>N</td>
<td>[SYNCHRONOUS IDLE]</td>
<td>53</td>
<td>35</td>
<td>5</td>
</tr>
<tr>
<td>23</td>
<td>O</td>
<td>[ENG OF TRANS. BLOCK]</td>
<td>54</td>
<td>36</td>
<td>6</td>
</tr>
<tr>
<td>24</td>
<td>P</td>
<td>[CANCEL]</td>
<td>55</td>
<td>37</td>
<td>7</td>
</tr>
<tr>
<td>25</td>
<td>Q</td>
<td>[END OF MEDIUM]</td>
<td>56</td>
<td>38</td>
<td>8</td>
</tr>
<tr>
<td>26</td>
<td>R</td>
<td>[END OF RECORD]</td>
<td>57</td>
<td>39</td>
<td>9</td>
</tr>
<tr>
<td>27</td>
<td>S</td>
<td>[END OF SELECT]</td>
<td>58</td>
<td>3A</td>
<td>;</td>
</tr>
<tr>
<td>28</td>
<td>T</td>
<td>[END OF TRANSMISSION]</td>
<td>59</td>
<td>3B</td>
<td>]</td>
</tr>
<tr>
<td>29</td>
<td>U</td>
<td>[END OF BINARY]</td>
<td>60</td>
<td>3C</td>
<td>^</td>
</tr>
<tr>
<td>30</td>
<td>V</td>
<td>[END OF TEXT]</td>
<td>61</td>
<td>3D</td>
<td>_</td>
</tr>
<tr>
<td>31</td>
<td>W</td>
<td>[END OF TRANSMISSION]</td>
<td>62</td>
<td>3E</td>
<td>`</td>
</tr>
<tr>
<td>32</td>
<td>X</td>
<td>[END OF SELECT]</td>
<td>63</td>
<td>3F</td>
<td>€</td>
</tr>
</tbody>
</table>

**Note:** This is `\n`: it's just another character!
That’s how `str` works.

- What about `float`?

- It’s harder to write 4.3752 as a sum of powers of two.
That’s how str works.
That’s how `str` works.

- Floating-point numbers are stored similarly to scientific notation:
That’s how \texttt{str} works.

- Floating-point numbers are stored similarly to scientific notation: \[ 1399.94 = 1.39994 \times 10^3 \]
That's how `str` works.

- Floating-point numbers are stored similarly to scientific notation: \(1399.94 = 1.39994 \times 10^3\)

- Need to store the base **and** the exponent. In memory, it looks something like this:
That’s how \texttt{str} works.

- Floating-point numbers are stored similarly to scientific notation: \(1399.94 = 1.39994 \times 10^3\)

- Need to store the base \textbf{and} the exponent. In memory, it looks something like this:
That's how \texttt{str} works.

- Floating-point numbers are stored similarly to scientific notation: \(1399.94 = 1.39994 \times 10^3\)
- Need to store the base \texttt{and} the exponent. In memory, it looks something like this:
That’s how $\texttt{str}$ works.

- Floating-point numbers are stored similarly to scientific notation: $1399.94 = 1.39994 \times 10^3$

- Need to store the base and the exponent. In memory, it looks something like this:
That’s how \texttt{str} works.

- Floating-point numbers are stored similarly to scientific notation: \(1399.94 = 1.39994 \times 10^3\)

- Need to store the base \textbf{and} the exponent. In memory, it looks something like this:

- Base and exponent are represented as base-2 integers, so the precision is finite: not all numbers can be represented!
Exercises

• Convert 1010101 to decimal.

• Convert 1023 to binary.
Next week

Making decisions:

if statements and boolean logic.