CSCI 141

Lecture 5:
More on print and input
Operator Precedence
Binary representation
Announcements

- Academic Honesty and googling for answers:
  - Searching the internet to learn about Python features, syntax, etc. **does not** violate academic honesty.
  - Programmers do this all the time.
  - You learned how to solve a problem!
  - Searching the internet for a solution to a problem I’ve given you and copy/pasting code **does** violate academic honesty.
  - You didn’t learn how to solve the problem.
Goals

• Know how to use *keyword arguments* such as the `sep` and `end` keyword arguments to the `print` function.

• Know how to save a function’s return value to a variable.

• Understand how the `+` operator behaves 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.
What have we covered so far?

- Data is (somehow) stored in memory.

  more on this today: representing numbers in binary!

- Each piece of data has a type.

  so far we’ve seen: int, float, str

- Variables can assign names to pieces of data.

  the assignment operator stores a value in a variable, as in:
  my_var = “Hello, world!”

- Operators can do things to the data (these operations are performed by the CPU).

  so far: assignment operator (=)
  arithmetic operators: (+, −, *, /, **, //, %)
What have we covered so far?

• A function can take inputs (arguments) and can produce an output (return value)
  
  so far: input, print, type, int, float, str

• Statements are instructions that are executed
  
  so far: assignment statements, such as my_var = 64 + 8

• Expressions are like phrases that can be evaluated to determine what value they represent.
  
  so far:
  • functions that return values, like int(42.8)
  • arithmetic expressions, like (4 + 2) / 2
  • and combinations of other expressions, like (2**3) // int(user_input)
Today’s Quiz

• Please write your name at the top:
  Lastname, Firstname

• 4 minutes
Today’s Quiz

• Please write your name at the top: Lastname, Firstname

• 4 minutes

• Working with a neighbor: do your answers agree? (2 minutes)
Function Calls: Getting Fancier

Syntax for a function call:

```
print("I am", 32, "years old")
```

- Function name
- Comma-separated list of arguments
- Open paren
- Close paren
Function Calls: Getting Fancier

Keyword arguments provide a way to pass optional arguments:

```
print("I am", 31, "years old", sep="\" \"")
```

sep keyword argument

The `print` function can take two keyword arguments:

- `sep` specifies what goes between the printed arguments (defaults to `sep=“ ”`)
- `end` specifies what goes after the last printed argument (defaults to `end=“\n”, the character representing a newline`)
input’s Return Value

The input function waits for the user to enter input on the keyboard:
```
input("Enter some input: ")
```

What if we want to store the input? Use a variable:
```
user_text = input("Enter some input: ")
```

input’s return value is whatever text the user entered

**Important**: input’s return value is always returns type str
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
Demo
Demo

- print with sep keyword arg
- print with end keyword arg
- save input and convert to an int

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"
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:

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

\[
\text{result} = 5 \% 3 \ ** \ 6 \ // \ 4
\]
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
We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: operator precedence.

Remember PEMDAS? BIDMAS? BODMAS?

Example:

10 * 6 ** 2 / 5 // 11 - 4
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:

```
10 * 6 ** 2 / 5 // 11 - 4
```
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?

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

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

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: \[ 104 = 1 \times 10^2 + 0 \times 10^1 + 4 \times 10^0 \] (hundreds place, tens place, ones place)

• The decimal representation of a number is a sum of multiples of the powers of ten.
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: $104 = 1 \times 10^2 + 0 \times 10^1 + 4 \times 10^0$ (hundreds place, tens place, 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.

• 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

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

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

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

The binary representation of the decimal number 23 is:

A. 10111
B. 11101
C. 01100
D. 11110
That’s how \texttt{int} works.

- What about \texttt{str} and \texttt{float}?
How do you store strings?

Various conventions exist:
- ASCII, Unicode

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?

**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>(BELL)</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>10</td>
<td>A</td>
<td>(LINE FEED)</td>
<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>
</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>
<tr>
<td>62</td>
<td>3E</td>
<td>?</td>
<td>94</td>
<td>5E</td>
<td>^</td>
<td>126</td>
<td>7E</td>
<td>_</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>3F</td>
<td>`</td>
<td>95</td>
<td>5F</td>
<td></td>
<td>127</td>
<td>7F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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.

- 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:

- 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.