Computational Basics

Distinguishing Data
Recap
Recap

-go.illinois.edu/cs101

- Late adds need to see the FAQ
- i>clickers start 1/30
- hw00, hw01 due 1/30
x = 10
y = x
x = 5

What is the value of y?
A 10
B 5
\[ \begin{align*}
  x &= 10 \\
  y &= x \\
  x &= 5
\end{align*} \]

What is the value of \( y \)?

A 10 ★

B 5
Components

- Literals – 4
Components

- **Literals** – 4
- **Operators** – +

Expressions – 4 + x

Statements – y = 4 + x
Components

- **Literals** — 4
- **Operators** — +
- **Variables** — x
Components

- Literals: 4
- Operators: +
- Variables: x
- Keywords: for

Expressions: 4 + x
Statements: y = 4 + x
Components

- Literals – 4
- Operators – +
- Variables – x
- Keywords – for
- Expressions – 4 + x
Components

- **Literals** – 4
- **Operators** – +
- **Variables** – \( x \)
- **Keywords** – for
- **Expressions** – \( 4 + x \)
- **Statements** – \( y = 4 + x \)
x = 10
How Assignment Works

\[
\begin{align*}
x &= 10 \\
y &= x \times x
\end{align*}
\]
How Assignment Works

\[
\begin{align*}
x & = 10 \\
y & = x \times x \\
x \times x & = y
\end{align*}
\]
How Assignment Works

```python
x = 10
y = x * x
x * x = y

x,y = y,x  # a neat trick to swap two variables
```
Data Types
$\mathbb{N}$

$1, 2, 3, 4, 5, \ldots$

natural numbers
$\mathbb{N}_0$

0, 1, 2, 3, 4, 5, ...

whole numbers
\[ \mathbb{Z} \]

..., −4, −3, −2, −1, 0, +1, +2, +3, ...

integers
Numbers

Q

rational numbers
...,-\frac{1}{4},-\frac{1}{5},-\frac{1}{6},0,\frac{1}{3},\frac{2}{3},\frac{10}{1},0.25,...

rational numbers
\[ \mathbb{R} \]

\[
\pi, \; , \; 10^{100}, \; +\frac{1}{10}, \; 0.25, \; ... 
\]

real numbers
complex numbers
Numbers

\[ \mathbb{C} \]

\[ \mathbb{C}, 1 + \mathbb{C}, \ldots \]

complex numbers
Python supports several basic number types:

- integer
- float
- complex
Python supports several basic number types:

- integer → $\mathbb{Z}$
- float → $\mathbb{R}$ or maybe $\mathbb{Q}$
- complex → $\mathbb{C}$ (again, maybe)
Floating-point numbers include a fractional part. (Anything with a decimal point—2.4, 3.0.)

complex is two floats together.

0 + 1j  # "i"
1 + 0j  # "1"
How do binary numbers work?

- Numeric types can be represented in binary:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
<td>100</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>101</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>2</td>
<td>110</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>3</td>
<td>111</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>
How do binary numbers work?

- Numeric types can be represented in binary:
  - 000 0 100 4
  - 001 1 101 5
  - 010 2 110 6
  - 011 3 111 7

- Basically, in decimal:

\[ 513 = 5 \times 10^2 + 1 \times 10^1 + 3 \times 10^0 \]
How do binary numbers work?

- Numeric types can be represented in binary:
  - 000 0 100 4
  - 001 1 101 5
  - 010 2 110 6
  - 011 3 111 7

- Basically, in decimal:
  \[ 513 = 5 \times 10^2 + 1 \times 10^1 + 3 \times 10^0 \]

- Similarly, in binary:
  \[ 1011_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \]
How do binary numbers work?

- Numeric types can be represented in binary:
  - 000 0 100 4
  - 001 1 101 5
  - 010 2 110 6
  - 011 3 111 7

- Basically, in decimal:
  \[ 513 = 5 \times 10^2 + 1 \times 10^1 + 3 \times 10^0 \]

- Similarly, in binary:
  \[ 1011_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \]
  \[ = 8 + 0 + 2 + 1 = 11 \]
How do binary numbers work?

- But there are only so many bytes in a number!

\[ 11100110 00001111 10000111 11100111 \]
How do binary numbers work?

- But there are only so many bytes in a number!

  11100110 00001111 10000111 11100111

- This represents a limit on the representation of certain numbers.
- If we add too much, the number may overflow.
Python integers can be arbitrarily large.
How do binary numbers work?

- Python integers can be arbitrarily large.
  
  10 ** 100
  10 ** (10 ** 5)
How do binary numbers work?

- Python integers can be arbitrarily large.
  \[ 10^{\text{100}} \]
  \[ 10^{(10^{5})} \]
- Floating-point numbers have limits, though.
  \[ 1.0 \times 10^{300} \] # okay
  \[ 1.0 \times 10^{340} \] # "infinite"
  \[ 1.0 \times 10^{-400} \] # "zero"
How do binary numbers work?

- Python integers can be arbitrarily large.
  
  \[
  10 \times 10^{100} \\
  10 \times (10 \times 10^5)
  \]

- Floating-point numbers have limits, though.
  
  \[
  1.0 \times 10^{300} \quad \# \text{ okay} \\
  1.0 \times 10^{340} \quad \# \text{ "infinite"} \\
  1.0 \times 10^{340} \quad \# \text{ "zero"}
  \]
How do binary numbers work?

- Python integers can be arbitrarily large.
  
  \[
  10 \times 10^{100} \\
  10^{10 \times (10^5)}
  \]

- Floating-point numbers have limits, though.
  
  \[
  1.0 \times 10^{300} \quad \text{# okay} \\
  1.0 \times 10^{340} \quad \text{# “infinite”} \\
  1.0 \times 10^{-400} \quad \text{# “zero”}
  \]
Floating-point numbers include a fractional part.
(Anything with a decimal point—2.4, 3.0.)

What are limits?
- Overflow/underflow (too big or too small)
- Arbitrary precision ($\pi, e$)
If one type is inadequate for a result, Python "promotes" the result.

```
1 / 3  # int => float
(-1) ** 0.5  # int => complex
(-1.0) ** 0.5 # float => complex
# why not -1 ** 0.5 to complex?
```
x = 4
y = 3 + 1j
z = 33.3333
print( x + y + z )

What is printed to the screen?
x = 4
y = 3 + 1j
z = 33.3333
print( x + y + z )

What is printed to the screen?

A  40
B  40.3333
C  40.3333 + 1j
D  None of the above
x = 4
y = 3 + 1j
z = 33.3333
print(x + y + z)

What is printed to the screen?
```python
x = 4
y = 3 + 1j
z = 33.3333
print( x + y + z )
```

What is printed to the screen?

A 40
B 40.3333
C 40.3333 + 1j
D None of the above
Our execution model

00111101 11101111 000000010

memory

11101111

x = 15

processor

15
What is an **encoding**?

01001000 01000101 01001100 01001100

What does a binary data value like this represent?

- What does binary data represent?
- How does the processor know?
What is an **encoding**?

01001000 01000101 01001100 01001100

What does a binary data value like this represent?

- What does binary data represent?
- How does the processor know?
- The **encoding** interprets the value.
What is a **data type**?

- A **data type** defines an encoding rule.
- All values have a type.
What is a **data type**?

- A **data type** defines an encoding rule.
- All values have a type.
- The type defines how data is represented in memory.
What is a **data type**?

- A **data type** defines an encoding rule.
- All values have a type.
- The type defines how data is represented in memory.
- The type defines allowed operations and how they work.
Operators
Evaluating an expression of integers will generally result in an integer answer.

- $3 + 5$
Evaluating an expression of integers will generally result in an integer answer

- $3 + 5$
- EXCEPTION: DIVISION!
Evaluating an expression of integers will generally result in an integer answer

- \( 3 + 5 \)
- **EXCEPTION: DIVISION!**
- \( 3 / 4 \rightarrow 0.75 \)
- Evaluating an expression of integers will generally result in an integer answer
  - $3 + 5$
  - **EXCEPTION: DIVISION!**
  - $3 \div 4 \rightarrow 0.75$
  - $3 \, // \, 4 \rightarrow 0$ (floor division)
Floating-point operations

- Evaluating an expression of floating-point values will result in a floating-point answer.
Floating-point operations

- Evaluating an expression of floating-point values will result in a floating-point answer.
  - $3.0 + 5.5 \rightarrow 8.5$

Engineers and scientists need to think carefully about the precision of answers.
Floating-point operations

- Evaluating an expression of floating-point values will result in a floating-point answer.
  - $3.0 + 5.5 \rightarrow 8.5$
  - $3.0 + 5.0 \rightarrow 8.0$

Engineers and scientists need to think carefully about the precision of answers.
Floating-point operations

- Evaluating an expression of floating-point values will result in a floating-point answer.
  - 3.0 + 5.5 $\rightarrow$ 8.5
  - 3.0 + 5.0 $\rightarrow$ 8.0
  - 3 + 5.5 $\rightarrow$ ? (what happens here?)
Floating-point operations

- Evaluating an expression of floating-point values will result in a floating-point answer.
  - \(3.0 + 5.5 \rightarrow 8.5\)
  - \(3.0 + 5.0 \rightarrow 8.0\)
  - \(3 + 5.5 \rightarrow ?\) (what happens here?)

- Engineers and scientists need to think carefully about the precision of answers.
Reaches inside of a value to access part of its data (called an attribute).
Reaches inside of a value to access part of its data (called an attribute).

Extracts special variables stored “inside” of the type.

```python
print(x.real)
print(x.imag)
```
Reaches inside of a value to access part of its data (called an attribute).
Extracts special variables stored “inside” of the type.

```python
print(x.real)
print(x.imag)
```
Both of these components are floats.
Question

\[ x = (3.5 + 1j) \]
\[ y = 1 \]
\[ z = x + y \]

What is the value of \( z.\text{imag} \)?
x = (3.5 + 1j)
y = 1
z = x + y

What is the value of z.imag?
A 4.5 + 1j
B 4.5
C 1j
D 1.0
Question

x = (3.5 + 1j)
y = 1
z = x + y

What is the value of z.imag?
x = (3.5 + 1j)
y = 1
z = x + y

What is the value of z.imag?
A 4.5 + 1j
B 4.5
C 1j
D 1.0 ★
String Data Type
How does text work?

- Each symbol is stored individually, one byte long:

  01001000 72
  01000101 69
  01001100 76
  01001100 76
  01001111 79
### ASCII encoding table

<table>
<thead>
<tr>
<th>Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>(nul)</td>
</tr>
<tr>
<td>001</td>
<td>(soh)</td>
</tr>
<tr>
<td>002</td>
<td>(stx)</td>
</tr>
<tr>
<td>003</td>
<td>(etx)</td>
</tr>
<tr>
<td>004</td>
<td>(eot)</td>
</tr>
<tr>
<td>005</td>
<td>(enq)</td>
</tr>
<tr>
<td>006</td>
<td>(ack)</td>
</tr>
<tr>
<td>007</td>
<td>(bel)</td>
</tr>
<tr>
<td>008</td>
<td>(bs)</td>
</tr>
<tr>
<td>009</td>
<td>(tab)</td>
</tr>
<tr>
<td>010</td>
<td>(lf)</td>
</tr>
<tr>
<td>011</td>
<td>(vt)</td>
</tr>
<tr>
<td>012</td>
<td>(nf)</td>
</tr>
<tr>
<td>013</td>
<td>(cr)</td>
</tr>
<tr>
<td>014</td>
<td>(so)</td>
</tr>
<tr>
<td>015</td>
<td>(si)</td>
</tr>
</tbody>
</table>

- **String Data Type**: HELLO

```
72 69 76 76 79
'HELLO'
```
### ASCII encoding table

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>016</td>
<td>(dle)</td>
</tr>
<tr>
<td>001</td>
<td>017</td>
<td>(dc1)</td>
</tr>
<tr>
<td>002</td>
<td>018</td>
<td>(dc2)</td>
</tr>
<tr>
<td>003</td>
<td>019</td>
<td>(dc3)</td>
</tr>
<tr>
<td>004</td>
<td>020</td>
<td>(dc4)</td>
</tr>
<tr>
<td>005</td>
<td>021</td>
<td>(nak)</td>
</tr>
<tr>
<td>006</td>
<td>022</td>
<td>(syn)</td>
</tr>
<tr>
<td>007</td>
<td>023</td>
<td>(etb)</td>
</tr>
<tr>
<td>008</td>
<td>024</td>
<td>(can)</td>
</tr>
<tr>
<td>009</td>
<td>025</td>
<td>(em)</td>
</tr>
<tr>
<td>010</td>
<td>026</td>
<td>(eof)</td>
</tr>
<tr>
<td>011</td>
<td>027</td>
<td>(esc)</td>
</tr>
<tr>
<td>012</td>
<td>028</td>
<td>(fs)</td>
</tr>
<tr>
<td>013</td>
<td>029</td>
<td>(gs)</td>
</tr>
<tr>
<td>014</td>
<td>030</td>
<td>(rs)</td>
</tr>
<tr>
<td>015</td>
<td>031</td>
<td>(us)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>064</td>
<td>@</td>
</tr>
<tr>
<td>065</td>
<td>A</td>
</tr>
<tr>
<td>066</td>
<td>B</td>
</tr>
<tr>
<td>067</td>
<td>C</td>
</tr>
<tr>
<td>068</td>
<td>D</td>
</tr>
<tr>
<td>069</td>
<td>E</td>
</tr>
<tr>
<td>070</td>
<td>F</td>
</tr>
<tr>
<td>071</td>
<td>G</td>
</tr>
<tr>
<td>072</td>
<td>H</td>
</tr>
<tr>
<td>073</td>
<td>I</td>
</tr>
<tr>
<td>074</td>
<td>J</td>
</tr>
<tr>
<td>075</td>
<td>K</td>
</tr>
<tr>
<td>076</td>
<td>L</td>
</tr>
<tr>
<td>077</td>
<td>M</td>
</tr>
<tr>
<td>078</td>
<td>N</td>
</tr>
<tr>
<td>079</td>
<td>O</td>
</tr>
</tbody>
</table>

72  69  76  76  79 = HELLO
### ASCII encoding table

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>(null)</td>
</tr>
<tr>
<td>001</td>
<td>(SOH)</td>
</tr>
<tr>
<td>002</td>
<td>(STX)</td>
</tr>
<tr>
<td>003</td>
<td>(ETX)</td>
</tr>
<tr>
<td>004</td>
<td>(EOT)</td>
</tr>
<tr>
<td>005</td>
<td>(ENQ)</td>
</tr>
<tr>
<td>006</td>
<td>(ACK)</td>
</tr>
<tr>
<td>007</td>
<td>(BEL)</td>
</tr>
<tr>
<td>008</td>
<td>(BS)</td>
</tr>
<tr>
<td>009</td>
<td>(HT)</td>
</tr>
<tr>
<td>010</td>
<td>(LF)</td>
</tr>
<tr>
<td>011</td>
<td>(VT)</td>
</tr>
<tr>
<td>012</td>
<td>(FF)</td>
</tr>
<tr>
<td>013</td>
<td>(CR)</td>
</tr>
<tr>
<td>014</td>
<td>(FS)</td>
</tr>
<tr>
<td>015</td>
<td>(GS)</td>
</tr>
<tr>
<td>016</td>
<td>(RS)</td>
</tr>
<tr>
<td>017</td>
<td>(US)</td>
</tr>
</tbody>
</table>

### Hexadecimal values

<table>
<thead>
<tr>
<th>Hex</th>
<th>Decimal</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>54</td>
<td>P</td>
</tr>
<tr>
<td>69</td>
<td>103</td>
<td>E</td>
</tr>
<tr>
<td>76</td>
<td>122</td>
<td>L</td>
</tr>
<tr>
<td>76</td>
<td>122</td>
<td>L</td>
</tr>
<tr>
<td>79</td>
<td>127</td>
<td>O</td>
</tr>
</tbody>
</table>

\[
72 \ 69 \ 76 \ 76 \ 79 = \text{HELLLO}'
\]
Strings

- As a literal: text surrounded by quotes (single or double).
  - "DEEP"
Strings

- As a literal: text surrounded by quotes (single or double).
  - "DEEP"
- Each symbol is a character.
Strings

- As a literal: text surrounded by quotes (single or double).
  - "DEEP"
- Each symbol is a character.
- Unlike numeric types, strings vary in length.
**String operations**

- **Concatenation**: combine two strings
  - Uses the + symbol
  - ‘RACE’ + ’CAR’
String operations

- **Concatenation**: combine two strings
  - Uses the + symbol
  - ’RACE’ + ’CAR’

- **Repetition**: repeat a string
  - Uses the *
  - ’HELLO ’*10
String operations

- **Concatenation**: combine two strings
  - Uses the + symbol
  - ’RACE’ + ’CAR’

- **Repetition**: repeat a string
  - Uses the *
  - ’HELLO ’*10

- **Formatting**: used to encode other data as string
  - Uses % symbol
Formatting operator

- Creates string with value inserted

```python
x = 100 * 54
s = "String is: %i" % x
print(s)
```
Formatting operator

- Creates string with value inserted
  - Formats nicely
  - Requires indicator of type inside of string

```python
x = 100 * 54
s = "String is: %i" % x
print(s)
```
Formatting operator

- Creates string with value inserted
  - Formats nicely
  - Requires indicator of type inside of string

```python
x = 100 * 54
s = "String is: %i" % x
print(s)
```
```python
name = "Neal"
grade = 2 / 3
m1 = "Hello, %s!" % name  # %s = string
m2 = "Your grade is: %f." % grade  # %f = float
print(m1)
print(m2)
```
Example

```python
name = "Neal"
grade = 2 / 3
m1 = "Hello, %s!" % name  # %s = string
m2 = "Your grade is:  %f." % grade  # %f = float
print(m1)
print(m2)

Hello, Neal!
Your grade is 0.66667.
```
\[ x = 3 \\
\text{s} = \left(\text{"%i"} \ % \ (x+1)\right) \ * \ x**(5\%x) \\
\text{print(s)} \]

What does this program print?

A 333333333333
B 444444444
C 9999
D %i%i%i%i%i
```python
x = 3
s = ("%i" % (x+1)) * x**(5%x)
print(s)
```

What does this program print?

A 333333333333
B 4444444444 ★
C 9999
D %i%i%i%i%i
Indexing operator

- Extracts single character

```java
String a = "FIRE";
a[0] // The integer is the index. We count from zero!
```

If negative, counts down from end.
Indexing operator

- Extracts single character
  
a = "FIRE"
  
a[0]
Indexing operator

- Extracts single character
  
a = "FIRE"
  
a[0]

- The integer is the index.
- Extracts single character
  \[ a = "FIRE" \]
  \[ a[0] \]
- The integer is the index.
- We count from zero!
Indexing operator

- Extracts single character
  ```
a = "FIRE"
a[0]
  ```
- The integer is the index.
- **We count from zero!**
- If negative, counts down from end.
s = "ABCDE"
i = 3
x = s[i]

What is the value of x?
A ’A’
B ’B’
C ’C’
D ’D’
E ’E’
s = "ABCDE"
i = 3
x = s[i]

What is the value of x?
A 'A'
B 'B'
C 'C'
D 'D' *
E 'E'
s = "ABCDE"
i = 25 % 3
y = s[i]

What is the value of y?
A 'A'
B 'B'
C 'C'
D 'D'
E 'E'
s = "ABCDE"
i = (11 % 3) - 7
z = s[i]

What is the value of z?
A 'A'
B 'B' ☆
C 'C'
D 'D'
E 'E'
Next steps

- Register i>clicker on Compass (testing 1/30)
- quiz02 (due 1/26)
- Complete hw00, hw01 (due 1/30)
- Attend your first lab
- Read for the next class