## Mutable Values

Announcements

Objects
(Demo)

## Objects

Objects

- Objects represent information


## Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions


## Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, \& processes


## Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, \& processes
- A type of object is called a class; classes are first-class values in Python


## Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, \& processes
- A type of object is called a class; classes are first-class values in Python
- Object-oriented programming:


## Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, \& processes
- A type of object is called a class; classes are first-class values in Python
- Object-oriented programming:
- A metaphor for organizing large programs


## Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, \& processes
- A type of object is called a class; classes are first-class values in Python
- Object-oriented programming:
- A metaphor for organizing large programs
- Special syntax that can improve the composition of programs


## Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, \& processes
- A type of object is called a class; classes are first-class values in Python
- Object-oriented programming:
- A metaphor for organizing large programs
- Special syntax that can improve the composition of programs
- In Python, every value is an object


## Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, \& processes
- A type of object is called a class; classes are first-class values in Python
- Object-oriented programming:
- A metaphor for organizing large programs
- Special syntax that can improve the composition of programs
- In Python, every value is an object
- All objects have attributes


## Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, \& processes
- A type of object is called a class; classes are first-class values in Python
- Object-oriented programming:
- A metaphor for organizing large programs
- Special syntax that can improve the composition of programs
- In Python, every value is an object
- All objects have attributes
- A lot of data manipulation happens through object methods


## Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, \& processes
- A type of object is called a class; classes are first-class values in Python
- Object-oriented programming:
- A metaphor for organizing large programs
- Special syntax that can improve the composition of programs
- In Python, every value is an object
- All objects have attributes
- A lot of data manipulation happens through object methods
- Functions do one thing; objects do many related things


# Example: Strings 

## Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

|  | ASCII Code Chart |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 0 | NUL | SOH | STX | ETX | EOT | ENQ | ACK | BEL | BS | HT | LF | VT | FF | CR | S0 | SI |
| 1 | DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB | CAN | EM | SUB | ESC | FS | GS | RS | US |
| 2 |  | ! | " | \# | \$ | \% | \& | , | ( | ) | * | + | , | - | . | / |
| 3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 4 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | 0 |
| 5 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \} | ] | $\wedge$ | - |
| 6 | , | a | b | c | d | e | f | g | h | i | j | k | 1 | m | n | $\bigcirc$ |
| 7 | p | 9 | r | s | t | u | v | w | x | y | z | \{ | \| | \} | $\sim$ | DEL |

## Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

|  |  |  |  |  |  |  |  | SCI | Cod | de C | art |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|  | 0 | NUL | SOH | STX | ETX | EOT | ENQ | ACK | BEL | BS | HT | LF | VT | FF | CR | S0 | SI |
| $\stackrel{\sim}{\sim}$ | 1 | DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB | CAN | EM | SUB | ESC | FS | GS | RS | US |
| - | 2 |  | ! | " | \# | \$ | \% | \& | ' | ( | ) | * | + | , | - | - | / |
| m | 3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| ï | 4 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | 0 |
| 3 | 5 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \} | ] | $\wedge$ | - |
| 2 | 6 |  | a | b | c | d | e | f | g | h | i | j | k | 1 | m | n | $\bigcirc$ |
| $\infty$ | 7 | p | q | r | s | t | u | v | w | x | y | z | \{ | \| | \} | $\sim$ | DEL |

## Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

000
001
010
011
100
101
110
111

|  |  |  |  |  |  |  |  | I |  | - | rt |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|  | 0 | NUL | SOH | STX | ETX | EOT | ENQ | ACK | BEL | BS | HT | LF | VT | FF | CR | S0 | SI |
| $\stackrel{\sim}{\sim}$ | 1 | DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB | CAN | EM | SUB | ESC | FS | GS | RS | US |
|  | 2 |  | ! | " | \# | \$ | \% | \& | ' | $($ | ) | * | + | , | - | . | / |
| m | 3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
|  | 4 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | 0 |
| 3 | 5 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | $\backslash$ | ] | $\wedge$ | - |
|  | 6 |  | a | b | c | d | e | f | g | h | i | j | k | 1 | m | n | 0 |
| $\infty$ | 7 | p | q | r | 5 | t | u | v | W | x | y | z | \{ | 1 | \} | $\sim$ | DEL |

## Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

|  |  |  |  |  |  |  |  |  | SCII |  | de C | art |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 000 |  | 0 | NUL | SOH | STX | ETX | EOT | ENQ | ACK | BEL | BS | HT | LF | VT | FF | CR | SO | SI |
| 001 | ๑ | 1 | DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB | CAN | EM | SUB | ESC | FS | GS | RS | US |
| 010 | - | 2 |  | ! | " | \# | \$ | \% | \& | . | ( | ) | * | + | , | - | . | / |
| 011 | m | 3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 100 | u' | 4 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | 0 |
| 101 | 3 | 5 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | $\backslash$ | ] | $\wedge$ | - |
| 110 | , | 6 | - | a | b | c | d | e | f | g | h | i | j | k | 1 | m | n | $\bigcirc$ |
| 111 | $\infty$ | 7 | P | 9 | r | s | t | u | v | w | x | y | z | \{ | \| | \} | $\sim$ | DEL |

16 columns: 4 bits

## Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

000
001
010
011
100
101
110
111

|  |  |  |  |  |  |  |  | SCII |  | e | art |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|  | 0 | NUL | SOH | STX | ETX | EOT | ENQ | ACK | BEL | BS | HT | LF | VT | FF | CR | S0 | SI |
| ¢ | 1 | DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB | CAN | EM | SUB | ESC | FS | GS | RS | US |
|  | 2 |  | ! | " | \# | \$ | \% | \& | , | ( | ) | * | + | , | - | . | 1 |
| $m$ | 3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| ü | 4 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | 0 |
| 3 | 5 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \} | ] | $\wedge$ | - |
|  | 6 |  | a | b | c | d | e | f | g | h | i | j | k | 1 | m | n | 0 |
|  | 7 | p | 9 | r | S | t | u | v | w | x | y | z | \{ | I | \} | $\sim$ | DEL |

16 columns: 4 bits

- Layout was chosen to support sorting by character code


## Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

000
001
010
011
100
101
110
111


16 columns: 4 bits

- Layout was chosen to support sorting by character code
- Rows indexed 2-5 are a useful 6-bit (64 element) subset


## Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

000
001
010
011
100
101
110
111

|  |  |  |  |  |  |  |  | SCII |  | de C | art |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|  | 0 | NUL | SOH | STX | ETX | EOT | ENQ | ACK | BEL | BS | HT | LF | VT | FF | CR | S0 | SI |
| ヘ | 1 | DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB | CAN | EM | SUB | ESC | FS | GS | RS | US |
| - | 2 |  | ! | " | \# | \$ | \% | \& | , | 1 | ) | * | + | , | - | . | / |
| m | 3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| ï | 4 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | 0 |
| 3 | 5 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | $\backslash$ | ] | $\wedge$ | - |
|  | 6 | , | a | b | c | d | e | f | g | h | i | j | k | 1 | m | n | $\bigcirc$ |
| $\infty$ | 7 | p | 9 | r | s | t | u | v | W | $\mathbf{x}$ | y | z | \{ | \| | \} | $\sim$ | DEL |

16 columns: 4 bits

- Layout was chosen to support sorting by character code
- Rows indexed 2-5 are a useful 6-bit (64 element) subset
- Control characters were designed for transmission


## Representing Strings: the ASCII Standard

> American Standard Code for Information Interchange

000
001
010
011
100
101
110
111

|  |  |  |  |  |  |  |  | SCII | Cod | C | art |  |  | -ine | fee | $d^{\prime \prime}$ | ( $\backslash \mathrm{n}$ ) |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|  | 0 | NUL | SOH | STX | ETX | EOT | ENQ | ACK | BEL | BS | HT | LF | VT | FF | CR | SO | SI |
| ¢ | 1 | DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB | CAN | EM | SUB | ESC | FS | GS | RS | US |
| - | 2 |  | ! | " | \# | \$ | \% | \& | ' | ( | ) | * | + | , | - | . | / |
| m | 3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| ü | 4 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | 0 |
| 3 | 5 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | $\backslash$ | ] | $\wedge$ | - |
|  | 6 | , | a | b | c | d | e | f | g | h | i | j | k | 1 | m | n | $\bigcirc$ |
| $\infty$ | 7 | P | q | r | 5 | t | u | v | W | x | y | z | \{ | 1 | \} | $\sim$ | DEL |

16 columns: 4 bits

- Layout was chosen to support sorting by character code
- Rows indexed 2-5 are a useful 6-bit (64 element) subset
- Control characters were designed for transmission


## Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

000
001
010
011
100
101
110
111


16 columns: 4 bits

- Layout was chosen to support sorting by character code
- Rows indexed 2-5 are a useful 6-bit (64 element) subset
- Control characters were designed for transmission


## Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

000
001
010
011
100
101
110
111


16 columns: 4 bits

- Layout was chosen to support sorting by character code
- Rows indexed 2-5 are a useful 6-bit (64 element) subset
- Control characters were designed for transmission
(Demo)


## Representing Strings: the Unicode Standard

## Representing Strings：the Unicode Standard

| 䋕 <br> 8071 | 睪毎 <br> 8072 | 倁 | $\begin{gathered} \text { 耳南 } \\ 8074 \\ \hline \end{gathered}$ | 耳貴 <br> 8075 | 㫕 <br> 8076 | 職 <br> 8077 | 耳俗 $8078$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| ｜建 | 腲 | 朌䜪 |  | 服 ${ }_{8175}$ | 腶 | 腷 | 日易 |
| 乹 <br> 8271 | $\underset{8272}{\text { 分 }}$ | 丰色 $8273$ | 弗他 <br> 8274 | $\begin{gathered} \text { 并他 } \\ \hline 8275 \\ \hline \end{gathered}$ | $\begin{gathered} \text { 豊他 } \\ 8276 \\ \hline \end{gathered}$ | 豐他 $8277$ | 山山 <br> 8278 |
| 芼 <br> 8371 |  | $\begin{aligned} & \text { 吉 } \\ & \text { 8373 } \end{aligned}$ | 扶 <br> 8374 | $\begin{gathered} \text { 炙 } \\ \text { 分 } \end{gathered}$ | 枤 <br> 8376 | 何 <br> 8377 | 苦 $8378$ |
| 苞 | 㫤 | 蒇 | 葴 | 状 | 古 | 莗 | 苖 |

## Representing Strings：the Unicode Standard

－137，994 characters in Unicode 12.1

| 薙 | ${ }_{\text {䏿 }}^{\text {802 }}$ | ${ }_{\text {缶 }}^{\text {樶 }}$ | ｜聴 | ${ }_{\text {聵 }}^{8075}$ | $\underset{\text { 8078 }}{\text { 最 }}$ |  | 聆 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| ｜建 | 腲 | 腳 | 朓 | 腵 | 腶 | 腷 | 腸 |
| 881 | 812 | ${ }_{873}$ | 874 | 875 | 876 | 817 | 8178 |
| 䬶 | 色 | 艳 | 弗色 | 并色 | $\frac{\text { 豊色 }}{8776}$ | $\frac{\text { 酆色 }}{8277}$ | 幽 |
| 芼 | $\underset{832}{\text { 荲 }}$ | $\begin{aligned} & \text { 壹 } \\ & 8373 \end{aligned}$ | 苵 | 密 | $\underset{8}{\substack{\text { 茳 } \\ 8376}}$ | $\begin{array}{\|l\|l\|} \substack{\text { 荷 } \\ \hline \\ \hline} \end{array}$ | $\underset{\text { 838 }}{\substack{\text { 菜 }}}$ |
| 苞 | 㫤 | 葳 | 葴 | 葵 | 葶 | 莗 | 思 |

## Representing Strings：the Unicode Standard

－137，994 characters in Unicode 12.1
－ 150 scripts（organized）

| 驁 | $\underset{\text { 8072 }}{\text { 棨 }}$ | $\underset{\text { 年 }}{\text { 圱 }}$ | 聴 | $\underset{\text { 807 }}{\text { 聵 }}$ | $\underset{\text { 最 }}{\text { 最 }}$ | 職 | 溏 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| ｜建 | 腲 | 腳 | 胦 | 腵 | 腶 | 腷 | 腸 |
| 8817 | 812 | ${ }_{873}$ | 8774 | ${ }_{8175}$ | 876 | 817 | 878 |
| $\begin{aligned} & \text { 䬶 } \\ & 8827 \end{aligned}$ | $\underset{872}{\text { 色 }}$ | 艳 | 弗色 | 剏 | 艶 | $\underset{8277}{\substack{\text { 嚖色 }}}$ | $\underset{\substack{\text { 8278 }}}{\text { 耂 }}$ |
| 芼 | 荲 | $\frac{\text { 营 }}{8373}$ | 苵 | $\underset{835}{\substack{\text { 芽 } \\ \hline}}$ | $\begin{gathered} \text { 获 } \\ \text { B37 } \end{gathered}$ | 荷 | $\begin{array}{\|l\|l\|} \substack{\text { 莩 } \\ \hline 878} \end{array}$ |
| 苞 | 㫤 | 葳 | 葴 | 葵 | 葶 | 莗 | 葸 |

## Representing Strings：the Unicode Standard

－137，994 characters in Unicode 12.1
－ 150 scripts（organized）
－Enumeration of character properties， such as case

| 㛑 | 聲 | 從 | 聴 | 聵 | 聶 | 職 | 渃 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 健 | 腲 | 腳 | 㹸 | 腵 | 腶 | 腷 |  |
|  | 812 | 8 | ${ }_{814}$ | 8ris | 88 | ${ }_{\text {sim }}$ |  |
| 朝 | 色 | 艳 | 䍖 | 艵 | 艶 | 艶 | 㞨 |
| 芼 | 荲 | 营 | 苵 | 葱 | 荶 | 荷 | 莩 |
| 苞 | 㫤 | 葳 | 葴 | 葵 | 葶 | 車 |  |

## Representing Strings：the Unicode Standard

－137，994 characters in Unicode 12.1
－ 150 scripts（organized）
－Enumeration of character properties， such as case
－Supports bidirectional display order

| 㪍 | $\underset{\text { 8072 }}{\text { 棨 }}$ | $\underset{\text { 年 }}{\text { 缶 }}$ | 聴 | 聵 | ${ }_{\text {最最 }}^{\text {8076 }}$ | 職 | 溏 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 健 | 腲 | 腳 | 朓 | 腵 | 腶 | 腷 | 腸 |
| 881 | 8172 | 873 | 8174 | ${ }_{8175}$ | ${ }_{876}$ | ${ }_{817}$ | ${ }_{817}$ |
| 㢃 | 色 | 艳 | 弗色 | 并色 | 艶 | 艷 | 屾 |
| 芼 | $\frac{\text { 堇 }}{8322}$ | 营 | 苵 | 葱 | 荶 | 荷 | 莩 |
| 苞 | 㫤 | 葳 | 葴 | 葵 | 葶 | 莗 | 葸 |

## Representing Strings：the Unicode Standard

－137，994 characters in Unicode 12.1
－ 150 scripts（organized）
－Enumeration of character properties， such as case
－Supports bidirectional display order
－A canonical name for every character

| $\underset{8071}{\text { 獒 }}$ | $\underset{\text { 8072 }}{\text { 棨 }}$ | $\underset{\text { 鹤 }}{\text { 年 }}$ | 聴 | $\underset{\text { 807 }}{\text { 聵 }}$ | $\underset{\text { 最 }}{\text { 最 }}$ | 職 | 溏 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 健 | 腲 | 腳 | 胦 | 腵 | 腶 | 腷 | 腸 |
| 8817 | 812 | ${ }_{813}$ | 874 | ${ }_{8175}$ | 876 | 817 | 878 |
| $\underset{\substack{8771}}{\substack{\text { 郎 }}}$ | $\underset{872}{\text { 色 }}$ | 艳 | 弗色 | $\begin{aligned} & \text { 并色 } \\ & 8275 \end{aligned}$ | $\begin{aligned} & \text { 豊色 } \\ & 8276 \end{aligned}$ | $\frac{\text { 艷色 }}{8277}$ | $\underset{\text { 8278 }}{\substack{\text { 为 }}}$ |
| 劳 | 荲 | $\frac{\text { 营 }}{873}$ | 苵 | 葱 | 苳 | 荷 | $\underset{\text { 䒯 }}{\text { 莐 }}$ |
| 苞 | 㫤 | 葳 | 葴 | 葵 | 葶 | 莗 | 思 |

## Representing Strings：the Unicode Standard

－137，994 characters in Unicode 12.1
－ 150 scripts（organized）
－Enumeration of character properties， such as case
－Supports bidirectional display order
－A canonical name for every character

| 数 | $\|$䏿 <br> 807 |  | $\underset{\substack{\text { 聴 } \\ \text { 807 }}}{\text { d }}$ | $\underset{\text { 聵 }}{\text { 805 }}$ | $\underset{\text { 最 }}{\text { 最 }}$ | 職 | 聸 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| ｜建 | 腲 | 腳 | 胦 | 腵 | 腶 | 腷 | 腸 |
| 881 | 812 | 873 | 817 | 8175 | ${ }_{876}$ | 817 | ${ }_{8178}$ |
| 㨁䬶 | 色 | 艳 | 弗色 | 并色 | 艶 | 豐色 | 州 |
| 荤 | $\frac{\text { 堇 }}{832}$ | $\begin{aligned} & \text { 营 } \\ & 8333 \end{aligned}$ | 苵 <br> ${ }^{8374}$ | $\underset{\substack{\text { 苞 } \\ \hline 875}}{ }$ | 获 | 荷 | 莩 |
| 葱 | 沓 | 葳 | 葴 | 癸 | 葶 | 莗 | 思 |

LATIN CAPITAL LETTER A

## Representing Strings：the Unicode Standard

－137，994 characters in Unicode 12.1
－ 150 scripts（organized）
－Enumeration of character properties， such as case
－Supports bidirectional display order
－A canonical name for every character

| 致 | 䏿 | 聳 | 聴 | 聵 | 最 | 職 | 聸 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 健 | 腲 | 腳 | 胦 | 腵 | 腶 | 腷 | 腸 |
| 8871 | 812 | 8173 | 8174 | 8175 | 876 | ${ }_{817}$ | ${ }_{878}$ |
| 㫰 | $\underset{872}{\text { 色 }}$ | 艳 | 弗色 <br> 8274 | 并色 | 豊色 | $\frac{\text { 艷色 }}{8277}$ | 岖 |
| 芼 | $\underset{832}{\text { 荲 }}$ | $\begin{aligned} & \text { 壹 } \\ & 8873 \end{aligned}$ | 苵 <br> 8374 | $\begin{array}{\|c} \text { 密 } \\ 8875 \end{array}$ | 荶 | 荷 | $\underset{\text { 838 }}{\substack{\text { 菜 }}}$ |
| 苞 | 㫤 | 葳 | 葴 | 葵 | 葶 | 莗 | 葸 |

LATIN CAPITAL LETTER A
DIE FACE－6

## Representing Strings：the Unicode Standard

－137，994 characters in Unicode 12.1
－ 150 scripts（organized）
－Enumeration of character properties， such as case
－Supports bidirectional display order
－A canonical name for every character

|  | ${ }_{\text {䏿 }}^{\text {等 }}$ | 慫 | 聴 | 聵 | $\underset{\text { 最最 }}{\text { 最 }}$ | 職 | $\underset{\substack{\text { 8078 } \\ \text { 儋 }}}{\text { den }}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 建 | 腲 | 腳 | 朓 | 腵 | 腶 | 腷 | 腸 |
| 8871 | 8172 | ${ }_{813}$ | 8174 | 8175 | 876 | ${ }_{817}$ | ${ }_{878}$ |
| 䬶 | $\underset{8272}{\text { 色 }}$ | 艳 | 弗色 | 并色 | 艶 | 酆色 | $\underset{\text { 878 }}{\text { 岖 }}$ |
| 芼 | 堇 | 营 | 荴 | 葱 | 获 | 荷 | 荸 |
| 837 | 332 | 873 | 8374 | ${ }_{875}$ | 836 | 837 | 838 |
| 苞 | 㫤 | 葳 | 葴 | 葵 | 葶 | 莗 | 思 |

LATIN CAPITAL LETTER A
DIE FACE－6

EIGHTH NOTE

## Representing Strings：the Unicode Standard

－137，994 characters in Unicode 12.1
－ 150 scripts（organized）
－Enumeration of character properties， such as case
－Supports bidirectional display order
－A canonical name for every character

| $\underset{\text { 獒 }}{\text { 807 }}$ | $\underset{\text { 8072 }}{\substack{\text { 棨 }}}$ |  | ${ }^{\text {聴 }}$ |  | $\underset{\text { 最 } 807}{\text { 最 }}$ | 職 | $\underset{\substack{\text { 8078 } \\ \text { 溏 }}}{\text { den }}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| ｜建 | 腲 | 腳 | 脜 | 腵 | 腶 | 腷 | 腸 |
| 881 | 812 | 873 | 8174 | 8175 | 8176 | 817 | ${ }_{878}$ |
| 䬶 <br> 8271 | $\underset{8272}{ }$ | 艳 | 弗色 | 并色 | $\begin{array}{\|l\|} \hline \text { 豊色 } \\ 8776 \end{array}$ | $\begin{aligned} & \text { 艷 } \\ & 8277 \end{aligned}$ | 岖 |
| 菫 | 荲 | $\begin{aligned} & \text { 营 } \\ & 8873 \end{aligned}$ | 苵 | $\underset{8375}{\substack{\text { 密 } \\ \hline}}$ | $\begin{aligned} & \text { 获 } \\ & \text { B77 } \end{aligned}$ | 荷菏 | $\underset{\text { 838 }}{\substack{\text { 莩 }}}$ |
| 葱 | 㫤 | 葳 | 葴 | 葵 | 葶 | 莗 | 葸 |

＇国＇

EIGHTH NOTE

## Representing Strings：the Unicode Standard

－137，994 characters in Unicode 12.1
－ 150 scripts（organized）
－Enumeration of character properties， such as case
－Supports bidirectional display order
－A canonical name for every character

LATIN CAPITAL LETTER A
DIE FACE－6

EIGHTH NOTE

| $\underset{\text { 㛑 }}{\text { 807 }}$ | $\underset{\text { 8072 }}{\text { 䈅 }}$ |  | $\left.\right\|_{\text {聴 }} ^{\text {8074 }}$ | ${ }_{\substack{\text { 聵 } \\ 807}}^{\text {d }}$ | $\underset{\text { 最最 }}{\text { 最 }}$ | 職 |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| ｜建 | 腲 | 腳 | 腉 | 腵 | 腶 | 腷 | 腸 |
| 871 | 812 | ${ }_{817}$ | 8174 | 8175 | 8176 | 817 | 818 |
| 䬶 <br> 8271 | $\underset{822}{\text { 色 }}$ | $\underset{827}{\substack{\text { 丰色 } \\ \hline}}$ | $\underset{8274}{\text { 弗色 }}$ | 并色 | 艶 | 艷 | 纠 |
| 芼 | 堇 | 营 | 苵 | 葱 | 荶 | 荷 | $\underset{\substack{\text { 838 }}}{\text { 荸 }}$ |
| 葱 | 㫤 | 葳 | 葴 | 葵 | 葶 | 莗 | 葸 |

＇国＇


## Representing Strings：the Unicode Standard

－137，994 characters in Unicode 12.1
－ 150 scripts（organized）
－Enumeration of character properties， such as case
－Supports bidirectional display order
－A canonical name for every character

LATIN CAPITAL LETTER A
DIE FACE－6

EIGHTH NOTE

| $\underset{\text { 㛑 }}{\text { 807 }}$ | $\underset{\text { 8072 }}{\text { 䈅 }}$ |  | $\left.\right\|_{\text {聴 }} ^{\text {8074 }}$ | ${ }_{\substack{\text { 聵 } \\ 807}}^{\text {d }}$ | $\underset{\text { 最最 }}{\text { 最 }}$ | 職 |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| ｜建 | 腲 | 腳 | 腉 | 腵 | 腶 | 腷 | 腸 |
| 871 | 812 | ${ }_{817}$ | 8174 | 8175 | 8176 | 817 | 818 |
| 䬶 <br> 8271 | $\underset{822}{\text { 色 }}$ | $\underset{827}{\substack{\text { 丰色 } \\ \hline}}$ | $\underset{8274}{\text { 弗色 }}$ | 并色 | 艶 | 艷 | 纠 |
| 芼 | 堇 | 营 | 苵 | 葱 | 荶 | 荷 | $\underset{\substack{\text { 838 }}}{\text { 荸 }}$ |
| 葱 | 㫤 | 葳 | 葴 | 葵 | 葶 | 莗 | 葸 |

＇国＇

（Demo）

Mutation Operations

## Some Objects Can Change

[Demo]

## Some Objects Can Change

[Demo]

First example in the course of an object changing state

## Some Objects Can Change

[Demo]

First example in the course of an object changing state
The same object can change in value throughout the course of computation

## Some Objects Can Change

[Demo]

First example in the course of an object changing state
The same object can change in value throughout the course of computation


## Some Objects Can Change

[Demo]

First example in the course of an object changing state
The same object can change in value throughout the course of computation


BABY

## Some Objects Can Change

[Demo]

First example in the course of an object changing state
The same object can change in value throughout the course of computation


## Some Objects Can Change

[Demo]

First example in the course of an object changing state
The same object can change in value throughout the course of computation


## Some Objects Can Change

[Demo]

First example in the course of an object changing state
The same object can change in value throughout the course of computation


## Some Objects Can Change

[Demo]

First example in the course of an object changing state
The same object can change in value throughout the course of computation


## Some Objects Can Change

[Demo]

First example in the course of an object changing state
The same object can change in value throughout the course of computation


## Some Objects Can Change

[Demo]

First example in the course of an object changing state
The same object can change in value throughout the course of computation


All names that refer to the same object are affected by a mutation

## Some Objects Can Change

[Demo]

First example in the course of an object changing state
The same object can change in value throughout the course of computation


All names that refer to the same object are affected by a mutation
Only objects of mutable types can change: lists \& dictionaries

## Some Objects Can Change

[Demo]

First example in the course of an object changing state
The same object can change in value throughout the course of computation


All names that refer to the same object are affected by a mutation Only objects of mutable types can change: lists \& dictionaries
\{Demo\}

## Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

## Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```
>>> four = [1, 2, 3, 4]
```


## Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
```


## Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
```


## Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2
```


## Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2
```


## Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2
```


## Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2
>>> four = [1, 2, 3, 4]
```


## Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2
>>> four = [1, 2, 3, 4]
>>> len(four)
4
```


## Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> another_mystery() # No arguments!
```


## Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> another_mystery() # No arguments!
>>> len(four)
2
```


## Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> another_mystery() # No arguments!
>>> len(four)
2
```

Tuples
(Demo)

Tuples are Immutable Sequences

## Tuples are Immutable Sequences

Immutable values are protected from mutation

## Tuples are Immutable Sequences

Immutable values are protected from mutation
>>> turtle $=(1,2,3)$

## Tuples are Immutable Sequences

Immutable values are protected from mutation
>>> turtle $=(1,2,3)$
>>> ooze()

## Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
```


## Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```


## Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```


## Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```


## Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
```


## Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```


## Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze() Next lecture: ooze can
>>> turtle = [1, 2, 3]
>>> turtle \begin{array}{l}{\mathrm{ Next lecture: ooze can}}\\{\mathrm{ change turtle's binding}}\end{array}]
>>> ooze()
(1, 2, 3) change turtle's binding
>>> turtle
['Anything could be inside!']
```


## Tuples are Immutable Sequences

Immutable values are protected from mutation
>> turtle $=(1,2,3)$
$\ggg$ ooze()
$\ggg$ turtle $\begin{aligned} & \text { Next lecture: ooze can } \\ & \text { change turtle's binding }\end{aligned}$
$(1,2,3)$
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']

The value of an expression can change because of changes in names or objects

## Tuples are Immutable Sequences

Immutable values are protected from mutation


The value of an expression can change because of changes in names or objects

Name change:

## Tuples are Immutable Sequences

Immutable values are protected from mutation


The value of an expression can change because of changes in names or objects

$$
\gg x+x
$$

Name change:

$$
\gg x+x
$$

## Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3) >>> turtle = [1, 2, 3]
>>> ooze() Next lecture: ooze can >>> ooze()
>>> turtle
```

>>> ooze()
>>> turtle
['Anything could be inside!']

The value of an expression can change because of changes in names or objects

$$
\begin{aligned}
& \ggg=2 \\
& \ggg \gg x+x
\end{aligned}
$$

Name change:

```
>> x + x
```


## Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3) >>> turtle = [1, 2, 3]
>>> ooze() Next lecture: ooze can >>> ooze()
>>> turtle
```

>>> ooze()
>>> turtle
['Anything could be inside!']

The value of an expression can change because of changes in names or objects

```
>>> x = 2
>>> x + x
4
```

Name change:
$\gg x+x$

## Tuples are Immutable Sequences

Immutable values are protected from mutation


The value of an expression can change because of changes in names or objects

|  | $\gg x=2$ |
| ---: | :--- |
|  | $\gg x+x$ |
| Name change: $\quad$ | 4 |
|  | $\gg x=3$ |
|  | $\gg x+x$ |

## Tuples are Immutable Sequences

Immutable values are protected from mutation


The value of an expression can change because of changes in names or objects

|  | $\ggg x=2$ |
| :--- | :--- |
| Name change: $\quad$ | $4>x+x$ |
|  | $\ggg \gg x=3$ |
|  | $\ggg>x+x$ |
|  | 6 |

## Tuples are Immutable Sequences

Immutable values are protected from mutation


The value of an expression can change because of changes in names or objects

|  | $\ggg=2$ |
| :--- | :--- |
|  | $\ggg x+x$ |
| Name change: $\quad$ | 4 |
|  | $\ggg \gg$ |
|  | $\ggg>$ |
|  | 6 |$\quad$ Object mutation:

## Tuples are Immutable Sequences

Immutable values are protected from mutation


The value of an expression can change because of changes in names or objects

| Name change: | >>> $x=2$ | Object mutation: |  |
| :---: | :---: | :---: | :---: |
|  | >>> $x+x$ |  | >>> $\mathrm{x}+\mathrm{x}$ |
|  | $\stackrel{4}{\gg x=3}$ |  |  |
|  | >>> $x+x$ |  | >>> $\mathrm{x}+\mathrm{x}$ |
|  |  |  |  |

## Tuples are Immutable Sequences

Immutable values are protected from mutation


The value of an expression can change because of changes in names or objects

| Name change: | >>> $x=2$ | Object mutation: | >>> $\mathrm{x}=$ [1 |
| :---: | :---: | :---: | :---: |
|  | >>> $\mathrm{x}+\mathrm{x}$ |  | >>> $\mathrm{x}+\mathrm{x}$ |
|  | $\begin{aligned} & 4 \\ & \ggg=3 \end{aligned}$ |  |  |
|  | >> $x+x$ |  | >>> $\mathrm{x}+\mathrm{x}$ |

## Tuples are Immutable Sequences

Immutable values are protected from mutation


The value of an expression can change because of changes in names or objects

| Name change: | >>> $x=2$ | Object mutation: | >>> $x=[1,2]$ |
| :---: | :---: | :---: | :---: |
|  | >>> $x+x$ |  | >> $x+x$ |
|  | $\stackrel{4}{\ggg x=3}$ |  | [1, 2, 1, 2] |
|  | $\xrightarrow{\ggg} \mathrm{x}+\mathrm{x}$ |  | >>> $x+x$ |

## Tuples are Immutable Sequences

Immutable values are protected from mutation


The value of an expression can change because of changes in names or objects

| Name change: | >>> $x=2$ | Object mutation: | $\rightarrow \gg x=[1,2]$ |
| :---: | :---: | :---: | :---: |
|  | >>> $\mathrm{x}+\mathrm{x}$ |  | >>> $\mathrm{x}+\mathrm{x}$ |
|  | $\stackrel{4}{\ggg} x=3$ |  | $\begin{aligned} & {[1,2,1,2]} \\ & \ggg \\ & \text { x.append (3) } \end{aligned}$ |
|  | >>> $\mathrm{x}+\mathrm{x}$ |  | >>> $\mathrm{x}+\mathrm{x}$ |

## Tuples are Immutable Sequences

Immutable values are protected from mutation


The value of an expression can change because of changes in names or objects


## Tuples are Immutable Sequences

Immutable values are protected from mutation


The value of an expression can change because of changes in names or objects


An immutable sequence may still change if it contains a mutable value as an element

## Tuples are Immutable Sequences

Immutable values are protected from mutation


The value of an expression can change because of changes in names or objects


An immutable sequence may still change if it contains a mutable value as an element >>> $s=([1,2], 3)$

## Tuples are Immutable Sequences

Immutable values are protected from mutation


The value of an expression can change because of changes in names or objects

| Name change: | >>> $x=2$ | Object mutation: | $\rightarrow \gg x=[1,2]$ |
| :---: | :---: | :---: | :---: |
|  | >> $\mathrm{x}+\mathrm{x}$ |  | $\begin{aligned} & \gg x+x \\ & {[1,2,1,2]} \end{aligned}$ |
|  | >>> $x=3$ |  | >>> x.append(3) |
|  | >>> x ( ${ }^{\text {r }}$ |  | $\begin{aligned} & \gg x+x \\ & {[1,2,3,1,2,} \end{aligned}$ |

An immutable sequence may still change if it contains a mutable value as an element

```
>>> s = ([1, 2], 3)
>>> s[0] = 4
```


## Tuples are Immutable Sequences

Immutable values are protected from mutation


The value of an expression can change because of changes in names or objects

| Name change: | >>> $x=2$ | Object mutation: | $\rightarrow \gg x=[1,2]$ |
| :---: | :---: | :---: | :---: |
|  | >> $\mathrm{x}+\mathrm{x}$ |  | $\begin{aligned} & \gg x+x \\ & {[1,2,1,2]} \end{aligned}$ |
|  | >>> $x=3$ |  | >>> x.append(3) |
|  | >>> x ( ${ }^{\text {r }}$ |  | $\begin{aligned} & \gg x+x \\ & {[1,2,3,1,2,} \end{aligned}$ |

An immutable sequence may still change if it contains a mutable value as an element

```
>>> s = ([1, 2], 3)
>>> s[0] = 4
ERROR
```


## Tuples are Immutable Sequences

Immutable values are protected from mutation


The value of an expression can change because of changes in names or objects

| Name change: | >>> $x=2$ | Object mutation: | $\rightarrow \gg x=[1,2]$ |
| :---: | :---: | :---: | :---: |
|  | >> $\mathrm{x}+\mathrm{x}$ |  | $\begin{aligned} & \gg x+x \\ & {[1,2,1,2]} \end{aligned}$ |
|  | >>> $x=3$ |  | >>> x.append(3) |
|  | >>> x ( ${ }^{\text {r }}$ |  | $\begin{aligned} & \gg x+x \\ & {[1,2,3,1,2,} \end{aligned}$ |

An immutable sequence may still change if it contains a mutable value as an element

```
>>> s = ([1, 2], 3)
>>> s = ([1, 2], 3)
```

>> s[0] = 4
ERROR

## Tuples are Immutable Sequences

Immutable values are protected from mutation


The value of an expression can change because of changes in names or objects

| Name change: | >>> $x=2$ | Object mutation: | $\rightarrow \gg x=[1,2]$ |
| :---: | :---: | :---: | :---: |
|  | >> $\mathrm{x}+\mathrm{x}$ |  | $\begin{aligned} & \gg x+x \\ & {[1,2,1,2]} \end{aligned}$ |
|  | >>> $x=3$ |  | >>> x.append(3) |
|  | >>> x ( ${ }^{\text {r }}$ |  | $\begin{aligned} & \gg x+x \\ & {[1,2,3,1,2,} \end{aligned}$ |

An immutable sequence may still change if it contains a mutable value as an element

```
>>> s = ([1, 2], 3)
>>> s[0] = 4
>>> s = ([1, 2], 3)
>>> s[0][0] = 4
```

ERROR

## Tuples are Immutable Sequences

Immutable values are protected from mutation


The value of an expression can change because of changes in names or objects

| Name change: | >>> $x=2$ | Object mutation: | $\rightarrow \gg x=[1,2]$ |
| :---: | :---: | :---: | :---: |
|  | >> $\mathrm{x}+\mathrm{x}$ |  | $\begin{aligned} & \gg x+x \\ & {[1,2,1,2]} \end{aligned}$ |
|  | >>> $x=3$ |  | >>> x.append(3) |
|  | >>> x ( ${ }^{\text {r }}$ |  | $\begin{aligned} & \gg x+x \\ & {[1,2,3,1,2,} \end{aligned}$ |

An immutable sequence may still change if it contains a mutable value as an element

```
>>> s = ([1, 2], 3)
>>> s[0] = 4
ERROR
```

```
>> s = ([1, 2], 3)
```

>> s = ([1, 2], 3)
>> s[0][0] = 4
>> s[0][0] = 4
>>> S

```
>>> S
```


## Tuples are Immutable Sequences

Immutable values are protected from mutation


The value of an expression can change because of changes in names or objects

| Name change: | >>> $x=2$ | Object mutation: | $\rightarrow \gg x=[1,2]$ |
| :---: | :---: | :---: | :---: |
|  | >> $\mathrm{x}+\mathrm{x}$ |  | $\begin{aligned} & \gg x+x \\ & {[1,2,1,2]} \end{aligned}$ |
|  | >>> $x=3$ |  | >>> x.append(3) |
|  | >>> x ( ${ }^{\text {r }}$ |  | $\begin{aligned} & \gg x+x \\ & {[1,2,3,1,2,} \end{aligned}$ |

An immutable sequence may still change if it contains a mutable value as an element

```
>>> s = ([1, 2], 3)
>>> s[0] = 4
ERROR
>>> s = ([1, 2], 3)
>> s[0][0] = 4
>>> S
([4, 2], 3)
```

Mutation

## Sameness and Change

## Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces


## Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator


## Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change


## Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed


## Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed - A list is still "the same" list even if we change its contents


## Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed - A list is still "the same" list even if we change its contents
>>> a = [10]


## Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed - A list is still "the same" list even if we change its contents

$$
\begin{aligned}
& \gg a=[10] \\
& \gg b=a
\end{aligned}
$$

## Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed - A list is still "the same" list even if we change its contents

$$
\begin{aligned}
& \ggg=[10] \\
& \gg b=a \\
& \ggg=b=b \\
& \text { True }
\end{aligned}
$$

## Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed - A list is still "the same" list even if we change its contents

```
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
```


## Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed - A list is still "the same" list even if we change its contents

```
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a
[10, 20]
```


## Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed - A list is still "the same" list even if we change its contents

```
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a
[10, 20]
>>> b
[10, 20]
```


## Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed - A list is still "the same" list even if we change its contents

```
>>> a = [10]
>> b = a
>>> a == b
True
>>> a.append(20)
>>> a
[10, 20]
>>> b
[10, 20]
>>> a == b
True
```


## Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed - A list is still "the same" list even if we change its contents
- Conversely, we could have two lists that happen to have the same contents, but are different

```
>>> a = [10]
>>> b = a
>> a == b
True
>>> a.append(20)
>>> a
[10, 20]
>>> b
[10, 20]
>>> a == b
True
```


## Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed - A list is still "the same" list even if we change its contents
- Conversely, we could have two lists that happen to have the same contents, but are different

```
>>> a = [10] >>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a
[10, 20]
>>> b
[10, 20]
>>> a == b
True
```


## Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed - A list is still "the same" list even if we change its contents
- Conversely, we could have two lists that happen to have the same contents, but are different

```
>>> a = [10] >>> a = [10]
>>> b = a >>> b = [10]
>>> a == b
True
>>> a.append(20)
>>> a
[10, 20]
>>> b
[10, 20]
>>> a == b
True
```


## Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed - A list is still "the same" list even if we change its contents
- Conversely, we could have two lists that happen to have the same contents, but are different

```
>>> a = [10] >>> a = [10]
>>> b = a >>> b = [10]
>> a == b >>> a == b
True True
>>> a.append(20)
>>> a
[10, 20]
>>> b
[10, 20]
>>> a == b
True
```


## Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed - A list is still "the same" list even if we change its contents
- Conversely, we could have two lists that happen to have the same contents, but are different

```
>>> a = [10] >>> a = [10]
>>> b = a >>> b = [10]
>> a == b >>> a == b
True
>>> a.append(20)
>>> a
[10, 20]
>>> b
[10, 20]
>>> a == b
True
```


## Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed - A list is still "the same" list even if we change its contents
- Conversely, we could have two lists that happen to have the same contents, but are different

```
>>> a = [10] >>> a = [10]
>>> b = a >>> b = [10]
>> a == b >>> a == b
True
>>> a.append(20)
>>> a
[10, 20]
>>> b
[10, 20]
>>> a == b
True
```


## Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed - A list is still "the same" list even if we change its contents
- Conversely, we could have two lists that happen to have the same contents, but are different

```
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a
[10, 20]
>>> b
[10, 20]
>>> a == b
True
```


## Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed - A list is still "the same" list even if we change its contents
- Conversely, we could have two lists that happen to have the same contents, but are different

```
>>> a = [10] >>> a = [10]
>>> b = a >>> b = [10]
>>> a == b
True
>>> a.append(20)
>>> a
[10, 20]
>>> b
[10, 20]
>>> a == b
True
```

```
>> a == b
```

>> a == b
True
True
>>> b.append(20)
>>> b.append(20)
>>> a
>>> a
[10]
[10]
>>> b
>>> b
[10, 20]
[10, 20]
>>> a == b
>>> a == b
False

```
False
```


## Identity Operators

## Identity Operators

## Identity

<exp0> is <exp1>
evaluates to True if both <exp0> and <exp1> evaluate to the same object

## Identity Operators

```
                    Identity
                            <exp0> is <exp1>
evaluates to True if both <exp0> and <exp1> evaluate to the same object
            Equality
    <exp0> == <exp1>
    evaluates to True if both <exp0> and <exp1> evaluate to equal values
```


## Identity Operators

```
                    Identity
                    <exp0> is <exp1>
evaluates to True if both <exp0> and <exp1> evaluate to the same object
                    Equality
    <exp0> == <exp1>
    evaluates to True if both <exp0> and <exp1> evaluate to equal values
    Identical objects are always equal values
```


## Identity Operators

```
    Identity
    <exp0> is <exp1>
evaluates to True if both <exp0> and <exp1> evaluate to the same object
    Equality
    <exp0> == <exp1>
    evaluates to True if both <exp0> and <exp1> evaluate to equal values
    Identical objects are always equal values
```

    (Demo)
    Mutable Default Arguments are Dangerous

## Mutable Default Arguments are Dangerous

A default argument value is part of a function value, not generated by a call

## Mutable Default Arguments are Dangerous

A default argument value is part of a function value, not generated by a call

```
>>> def f(s=[]):
*" s.append(3)
*". return len(s)
...
```


## Mutable Default Arguments are Dangerous

A default argument value is part of a function value, not generated by a call

```
>>> def f(s=[]):
*" s.append(3)
... return len(s)
>>> f()
1
```


## Mutable Default Arguments are Dangerous

A default argument value is part of a function value, not generated by a call

```
>>> def f(s=[]):
#." s.append(3)
#. return len(s)
>>> f()
1
>>> f()
2
```


## Mutable Default Arguments are Dangerous

A default argument value is part of a function value, not generated by a call

```
>>> def f(s=[]):
." s.append(3)
... return len(s)
>>> f()
1
>>> f()
2
>>> f()
3
```


## Mutable Default Arguments are Dangerous

A default argument value is part of a function value, not generated by a call

```
>>> def f(s=[]):
*" s.append(3)
... return len(s)
>>> f()
1
>>> f()
2
>>> f()
3
```



## Mutable Default Arguments are Dangerous

A default argument value is part of a function value, not generated by a call

```
>>> def f(s=[]):
*" s.append(3)
... return len(s)
>>> f()
1
>>> f()
2
>>> f()
3
```



Lists

Lists in Environment Diagrams

## Lists in Environment Diagrams

Assume that before each example below we execute:
$\mathrm{s}=[2,3]$
$t=[5,6]$

## Lists in Environment Diagrams

Assume that before each example below we execute:
$\mathrm{s}=[2,3]$
$t=[5,6]$

## Lists in Environment Diagrams

Assume that before each example below we execute:
$\mathrm{s}=[2,3]$
$t=[5,6]$

## Lists in Environment Diagrams

Assume that before each example below we execute:
$\mathrm{s}=[2,3]$
$t=[5,6]$

## Lists in Environment Diagrams

Assume that before each example below we execute:
$\mathrm{s}=[2,3]$
$t=[5,6]$

| Operation | Example | Result |
| :--- | :--- | :--- |
| append adds one <br> element to a list |  |  |
|  |  |  |

## Lists in Environment Diagrams

Assume that before each example below we execute:
$\mathrm{s}=[2,3]$
$t=[5,6]$

| Operation | Example | Result |
| :--- | :--- | :--- |
| append adds one <br> element to a list | S.append(t) <br> $t=0$ |  |
|  |  |  |

## Lists in Environment Diagrams

## Assume that before each example below we execute:

s = $[2,3]$
$t=[5,6]$

| Operation | Example | Result | Global |
| :--- | :--- | :--- | :--- |
| append adds one <br> element to a list | s.append(t) <br> $\mathrm{t}=0$ |  |  |

## Lists in Environment Diagrams

```
Assume that before each example below we execute:
\(\mathrm{s}=[2,3]\)
\(t=[5,6]\)
```

| Operation | Example | Result |
| :--- | :--- | :--- |
| append adds one <br> element to a list | s.append(t) <br> $\mathrm{t}=0$ |  |
|  |  |  |
|  |  |  |



## Lists in Environment Diagrams

```
Assume that before each example below we execute:
\(\mathrm{s}=[2,3]\)
\(t=[5,6]\)
```

| Operation | Example | Result |
| :--- | :--- | :--- |
| append adds one <br> element to a list | S.append(t) <br> $\mathrm{t}=0$ |  |
|  |  |  |



## Lists in Environment Diagrams

```
Assume that before each example below we execute:
\(\mathrm{s}=[2,3]\)
\(t=[5,6]\)
```

| Operation | Example | Result |
| :--- | :--- | :--- |
| append adds one <br> element to a list | s.append(t) <br> t $=0$ |  |
|  |  |  |



## Lists in Environment Diagrams

```
Assume that before each example below we execute:
\(\mathrm{s}=[2,3]\)
\(t=[5,6]\)
```



## Lists in Environment Diagrams

```
Assume that before each example below we execute:
\(s=[2,3]\)
\(t=[5,6]\)
```

| Operation | Example | Result | Global |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
| append adds one element to a list | $\begin{aligned} & \text { s.append(t) } \\ & t=0 \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3,[5,6]] \\ & t \rightarrow 0 \end{aligned}$ |  | list <br> 0 <br> 2 | ${ }^{1} 3$ |
|  |  |  |  | list |  |
|  |  |  |  | ${ }^{0} 5$ | $6$ |

## Lists in Environment Diagrams

```
Assume that before each example below we execute:
s = \([2,3]\)
\(t=[5,6]\)
```

| Operation | Example | Result | Global |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
| append adds one element to a list | $\begin{aligned} & \text { s.append (t) } \\ & \mathrm{t}=0 \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3,[5,6]] \\ & t \rightarrow 0 \end{aligned}$ |  | list <br> 0 <br> $\mathbf{2}$ | ${ }^{1} 3$ |
| extend adds all elements in one list to another list |  |  |  | list |  |
|  |  |  |  |  | ${ }^{1} 6$ |

## Lists in Environment Diagrams

```
Assume that before each example below we execute:
s = \([2,3]\)
\(t=[5,6]\)
```

| Operation | Example | Result | Global |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
| append adds one element to a list | $\begin{aligned} & \text { s.append (t) } \\ & \mathrm{t}=0 \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3,[5,6]] \\ & t \rightarrow 0 \end{aligned}$ |  | list <br> 0 <br> $\mathbf{2}$ | ${ }^{1} 3$ |
| extend adds all <br> elements in one list to another list | $\begin{aligned} & \text { s.extend }(t) \\ & \mathrm{t}[1]=0 \end{aligned}$ |  |  | list |  |
|  |  |  |  |  | ${ }^{1} 6$ |

## Lists in Environment Diagrams

```
Assume that before each example below we execute:
s = \([2,3]\)
\(t=[5,6]\)
```

| Operation | Example | Result | Global |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| append adds one element to a list | $\begin{aligned} & \text { s.append }(\mathrm{t}) \\ & \mathrm{t}=0 \end{aligned}$ | $\begin{array}{ll} s \rightarrow[2,3,[5,6]] \\ t \rightarrow 0 \end{array}$ |  | $\begin{aligned} & \text { list } \\ & \begin{array}{\|c} 0 \\ 2 \end{array} \end{aligned}$ | ${ }^{1} 3$ | ${ }^{2} 5$ | ${ }^{3} 6$ |
| extend adds all elements in one list to another list | $\begin{aligned} & \text { s.extend }(t) \\ & t[1]=0 \end{aligned}$ |  |  | list |  |  |  |
|  |  |  |  | ${ }^{0} 5$ | ${ }^{1} 6$ |  |  |

## Lists in Environment Diagrams

```
Assume that before each example below we execute:
s = \([2,3]\)
\(t=[5,6]\)
```

| Operation | Example | Result | Global |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| append adds one element to a list | $\begin{aligned} & \text { s.append (t) } \\ & \mathrm{t}=0 \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3,[5,6]] \\ & t \rightarrow 0 \end{aligned}$ |  | list <br> ${ }^{0} \mathbf{2}$ | ${ }^{1} 3$ | ${ }^{2} 5$ | ${ }^{3} 6$ |
| extend adds all elements in one list to another list | $\begin{aligned} & \text { s.extend }(t) \\ & \mathrm{t}[1]=0 \end{aligned}$ |  |  | list |  |  |  |
|  |  |  |  |  | ${ }^{1} \times 0$ |  |  |

## Lists in Environment Diagrams

## Assume that before each example below we execute: <br> s = $[2,3]$ <br> $t=[5,6]$

| Operation | Example | Result | Global <br> s <br> t |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| append adds one element to a list | $\begin{aligned} & \text { s. append }(t) \\ & t=0 \end{aligned}$ | $\begin{aligned} & \mathrm{s} \rightarrow[2,3,[5,6]] \\ & \mathrm{t} \rightarrow 0 \end{aligned}$ |  | list 0 | ${ }^{1} 3$ | ${ }^{2} 5$ | ${ }^{3} 6$ |
| extend adds all elements in one list to another list | $\begin{aligned} & \text { s.extend }(\mathrm{t}) \\ & \mathrm{t}[1]=0 \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3,5,6] \\ & t \rightarrow[5,0] \end{aligned}$ |  | t |  |  |  |
|  |  |  |  | ${ }^{0} 5$ | ${ }^{1} \times 0$ |  |  |

## Lists in Environment Diagrams

## Assume that before each example below we execute: <br> s = $[2,3]$ <br> $t=[5,6]$

| Operation | Example | Result | Global |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
| append adds one element to a list | $\begin{aligned} & \text { s.append (t) } \\ & \mathrm{t}=0 \end{aligned}$ | $\begin{aligned} & \mathrm{s} \rightarrow[2,3,[5,6]] \\ & \mathrm{t} \rightarrow 0 \end{aligned}$ |  | list <br> 0 <br> 2 | ${ }^{1} 3$ |
| extend adds all elements in one list to another list | $\begin{aligned} & \text { s.extend }(t) \\ & \text { t[1] }=0 \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3,5,6] \\ & t \rightarrow[5,0] \end{aligned}$ |  | list |  |
|  |  |  |  | ${ }^{0} 5$ | ${ }^{1} 6$ |

## Lists in Environment Diagrams

## Assume that before each example below we execute: <br> s = $[2,3]$ <br> $t=[5,6]$

| Operation | Example | Result | Global | list |
| :---: | :---: | :---: | :---: | :---: |
| append adds one element to a list | $\begin{aligned} & \text { s.append(t) } \\ & t=0 \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3,[5,6]] \\ & t \rightarrow 0 \end{aligned}$ |  | 3 |
| ```extend adds all elements in one list to another list``` | $\begin{aligned} & \text { s.extend (t) } \\ & \mathrm{t}[1]=0 \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3,5,6] \\ & t \rightarrow[5,0] \end{aligned}$ |  |  |
| addition \& slicing <br> create new lists containing existing elements |  |  |  | ${ }^{1} 6$ |

## Lists in Environment Diagrams

## Assume that before each example below we execute: <br> s = $[2,3]$ <br> $t=[5,6]$

| Operation | Example | Result | Global | list |
| :---: | :---: | :---: | :---: | :---: |
| append adds one element to a list | $\begin{aligned} & \text { s.append }(t) \\ & t=0 \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3,[5,6]] \\ & t \rightarrow 0 \end{aligned}$ |  | $3$ |
| ```extend adds all elements in one list to another list``` | $\begin{aligned} & \text { s.extend(t) } \\ & \mathrm{t}[1]=0 \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3,5,6] \\ & t \rightarrow[5,0] \end{aligned}$ |  |  |
| addition \& slicing create new lists containing existing elements | $\begin{aligned} & a=s+[t] \\ & b=a[1:] \\ & a[1]=9 \\ & b[1][1]=0 \end{aligned}$ |  |  | 6 |

## Lists in Environment Diagrams

## Assume that before each example below we execute: <br> s = $[2,3]$ <br> $t=[5,6]$

| Operation | Example | Result | Global |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| append adds one element to a list | $\begin{aligned} & \text { s.append }(t) \\ & t=0 \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3,[5,6]] \\ & t \rightarrow 0 \end{aligned}$ |  |  | list <br> 0 <br> 2 | 1 <br> 3 |
| ```extend adds all elements in one list to another list``` | $\begin{aligned} & \text { s.extend(t) } \\ & t[1]=0 \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3,5,6] \\ & t \rightarrow[5,0] \end{aligned}$ |  |  | list |  |
| addition \& slicing <br> create new lists <br> containing existing elements | $\begin{aligned} & a=s+[t] \\ & b=a[1:] \\ & a[1]=9 \\ & b[1][1]=0 \end{aligned}$ |  |  |  | ${ }^{0} 5$ | ${ }^{1} 6$ |

## Lists in Environment Diagrams

## Assume that before each example below we execute: <br> s = $[2,3]$ <br> $t=[5,6]$



## Lists in Environment Diagrams

## Assume that before each example below we execute: <br> s = $[2,3]$ <br> $t=[5,6]$



## Lists in Environment Diagrams

## Assume that before each example below we execute: <br> s = $[2,3]$ <br> $t=[5,6]$



## Lists in Environment Diagrams

## Assume that before each example below we execute: <br> s = $[2,3]$ <br> $t=[5,6]$



## Lists in Environment Diagrams

## Assume that before each example below we execute: <br> s = $[2,3]$ <br> $t=[5,6]$



## Lists in Environment Diagrams

## Assume that before each example below we execute: <br> s = $[2,3]$ <br> $t=[5,6]$



## Lists in Environment Diagrams

## Assume that before each example below we execute: <br> s = $[2,3]$ <br> $t=[5,6]$



## Lists in Environment Diagrams

## Assume that before each example below we execute: <br> $\mathrm{s}=[2,3]$ <br> $t=[5,6]$



## Lists in Environment Diagrams

## Assume that before each example below we execute: <br> s = $[2,3]$ <br> $t=[5,6]$



## Lists in Environment Diagrams

## Assume that before each example below we execute: <br> $\mathrm{s}=[2,3]$ <br> $t=[5,6]$

| Operation | Example | Result | Global |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
| append adds one element to a list | $\begin{aligned} & \mathrm{s} \text {.append }(\mathrm{t}) \\ & \mathrm{t}=0 \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3,[5,6]] \\ & t \rightarrow 0 \end{aligned}$ |  | $\begin{array}{r}\text { list } \\ \hline 0 \\ \hline 0\end{array}$ | ${ }^{1} 3$ |
| extend adds all elements in one list to another list | $\begin{aligned} & \text { s.extend }(t) \\ & \mathrm{t}[1]=0 \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3,5,6] \\ & t \rightarrow[5,0] \end{aligned}$ |  | list | $1$ |
| addition \& slicing create new lists containing existing elements | $\begin{aligned} & a=s+[t] \\ & b=a[1:] \\ & a[1]=9 \\ & b[1][1]=0 \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3] \\ & t \rightarrow[5,0] \\ & a \rightarrow[2,9,[5,0]] \\ & b \rightarrow[3,[5,0]] \end{aligned}$ |  | $\mathbf{2}$ <br> list <br> 0 | 3 |
| The list function also creates a new list containing existing elements | $\begin{aligned} & \mathrm{t}=\text { list(s) } \\ & \mathrm{s}[1]=0 \end{aligned}$ |  |  | $5$ | 6 |

## Lists in Environment Diagrams

## Assume that before each example below we execute: <br> s = $[2,3]$ <br> $t=[5,6]$

| Operation | Example | Result | Global |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
| append adds one element to a list | $\begin{aligned} & \text { s.append (t) } \\ & \mathrm{t}=0 \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3,[5,6]] \\ & t \rightarrow 0 \end{aligned}$ |  | list <br> 0 <br> 8 | ${ }^{1} 3$ |
| extend adds all elements in one list to another list | $\begin{aligned} & \text { s.extend }(t) \\ & \mathrm{t}[1]=0 \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3,5,6] \\ & t \rightarrow[5,0] \end{aligned}$ |  | list | $1$ |
| addition \& slicing create new lists containing existing elements | $\begin{aligned} & a=s+[t] \\ & b=a[1:] \\ & a[1]=9 \\ & b[1][1]=0 \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3] \\ & t \rightarrow[5,0] \\ & a \rightarrow[2,9,[5,0]] \\ & b \rightarrow[3,[5,0]] \end{aligned}$ |  | $2$ | 3 <br> 1 |
| The list function also creates a new list containing existing elements | $\begin{aligned} & \mathrm{t}=\text { list(s) } \\ & \mathrm{s}[1]=0 \end{aligned}$ |  |  | $5$ | $6$ |

## Lists in Environment Diagrams

## Assume that before each example below we execute: <br> s = $[2,3]$ <br> $t=[5,6]$

| Operation | Example | Result | $\begin{array}{c\|c\|c} \hline \text { Global } \\ s & - \\ t & - \end{array}$ |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
| append adds one element to a list | $\begin{aligned} & \text { s.append }(\mathrm{t}) \\ & \mathrm{t}=0 \end{aligned}$ | $\operatorname{sic}_{\substack{s \\ t \rightarrow 0}}[2,[5,6]]$ |  | list <br> 0 <br> 2 | ${ }^{1} 80$ |
| extend adds all elements in one list to another list | $\begin{aligned} & \text { s. extend }(t) \\ & \mathrm{t}[1]=0 \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3,5,6] \\ & t \rightarrow[5,0] \end{aligned}$ |  | list |  |
| addition \& slicing create new lists containing existing elements | $\begin{aligned} & a=s+[t] \\ & b=a[1:] \\ & a[1]=9 \\ & b[1][1]=0 \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3] \\ & t \rightarrow[5,0] \\ & a \rightarrow[2,9,[5,0]] \\ & b \rightarrow[3,[5,0]] \end{aligned}$ |  | 2 | 3 |
| The list function also creates a new list containing existing elements | $\begin{aligned} & \mathrm{t}=\operatorname{list}(\mathrm{s}) \\ & \mathrm{s}[1]=0 \end{aligned}$ |  |  | 5 | 6 |

## Lists in Environment Diagrams

## Assume that before each example below we execute: <br> $\mathrm{s}=[2,3]$ <br> $t=[5,6]$

| Operation | Example | Result | $\begin{array}{c\|c} \text { Global } \\ \mathrm{s} & - \\ \mathrm{t} & - \end{array}$ |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
| append adds one element to a list | $\begin{aligned} & \text { s.append }(\mathrm{t}) \\ & \mathrm{t}=0 \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3,[5,6]] \\ & t \rightarrow 0 \end{aligned}$ |  | ${ }^{\text {list }}$ | ${ }^{1} 80$ |
| extend adds all elements in one list to another list | $\begin{aligned} & \text { s. extend }(t) \\ & t[1]=0 \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3,5,6] \\ & t \rightarrow[5,0] \end{aligned}$ |  | list |  |
| addition \& slicing create new lists containing existing elements | $\begin{aligned} & a=s+[t] \\ & b=a[1:] \\ & a[1]=9 \\ & b[1][1]=0 \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3] \\ & t \rightarrow[5,0] \\ & a \rightarrow[2,9,[5,0]] \\ & b \rightarrow[3,[5,0]] \end{aligned}$ |  | 2 | 3 |
| The list function also creates a new list containing existing elements | $\begin{aligned} & \mathrm{t}=\operatorname{list}(\mathrm{s}) \\ & \mathrm{s}[1]=0 \end{aligned}$ | $\begin{array}{lll} s \rightarrow[2, & 0] \\ t \rightarrow[2, & 3] \end{array}$ |  | 5 | 6 |

## Lists in Environment Diagrams

## Assume that before each example below we execute: <br> s = $[2,3]$ <br> $t=[5,6]$



## Lists in Environment Diagrams

## Assume that before each example below we execute: <br> s = $[2,3]$ <br> $t=[5,6]$



## Lists in Environment Diagrams

## Assume that before each example below we execute: <br> s = $[2,3]$ <br> $t=[5,6]$



## Lists in Environment Diagrams

## Assume that before each example below we execute: <br> s = $[2,3]$ <br> $t=[5,6]$



## Lists in Environment Diagrams

## Assume that before each example below we execute: <br> s = $[2,3]$ <br> $t=[5,6]$



## Lists in Environment Diagrams

Assume that before each example below we execute:
$\mathrm{s}=[2,3]$
$t=[5,6]$

## Lists in Environment Diagrams

Assume that before each example below we execute:
s = $[2,3]$
$t=[5,6]$
Operation
Example
Result

## Lists in Environment Diagrams

Assume that before each example below we execute:
s = $[2,3]$
$t=[5,6]$

| Operation | Example | Result |
| :--- | :--- | :--- |

pop removes \& returns
the last element

## Lists in Environment Diagrams

Assume that before each example below we execute:
s = $[2,3]$
$t=[5,6]$

| Operation | Example | Result |
| :--- | :--- | :--- |
| pop removes \& returns <br> the last element | $\mathrm{t}=\mathrm{s.pop()}$ |  |
|  |  |  |
|  |  |  |

## Lists in Environment Diagrams

Assume that before each example below we execute:
s = $[2,3]$
$t=[5,6]$

| Operation | Example | Result |
| :--- | :--- | :--- |
| pop removes \& returns <br> the last element | $t=s . p o p()$ | $s \rightarrow[2]$ <br> $t \rightarrow 3$ |

## Lists in Environment Diagrams

Assume that before each example below we execute:
s = $[2,3]$
$t=[5,6]$

| Operation | Example | Result |
| :--- | :--- | :--- |
| pop removes \& returns <br> the last element | $\mathrm{t}=\mathrm{s} . \mathrm{pop}()$ | $\mathrm{s} \rightarrow$ [2] <br> $\mathrm{t} \rightarrow 3$ |

remove removes the
first element equal
to the argument

## Lists in Environment Diagrams

## Assume that before each example below we execute:

s = $[2,3]$
$t=[5,6]$

| Operation | Example | Result |
| :--- | :--- | :--- |
| pop removes \& returns <br> the last element | t = s.pop() | s $\rightarrow$ [2] <br> $t \rightarrow 3$ |
| remove removes the <br> first element equal <br> to the argument | t.extend(t) <br> t.remove(5) |  |
|  |  |  |
|  |  |  |

## Lists in Environment Diagrams

## Assume that before each example below we execute:

s = $[2,3]$
$t=[5,6]$

| Operation | Example | Result |
| :--- | :--- | :--- |
| pop removes \& returns <br> the last element | $\mathrm{t}=\mathrm{s.pop()}$ | $\mathrm{s} \rightarrow$ [2] <br> $\mathrm{t} \rightarrow 3$ |
| remove removes the <br> first element equal <br> to the argument | t.extend(t) <br> t.remove(5) | $\mathrm{s} \rightarrow[2, ~ 3]$ <br> $\mathrm{t} \rightarrow[6,5,6]$ |

## Lists in Environment Diagrams

Assume that before each example below we execute:
s = $[2,3]$
$t=[5,6]$

| Operation | Example | Result |
| :--- | :--- | :--- |
| pop removes \& returns <br> the last element | $\mathrm{t}=\mathrm{s.pop()}$ | $\mathrm{s} \rightarrow$ [2] <br> $\mathrm{t} \rightarrow 3$ |
| remove removes the <br> first element equal <br> to the argument | t.extend(t) <br> $\mathrm{t} . \operatorname{remove(5)}$ | $\mathrm{s} \rightarrow[2, ~ 3]$ <br> $\mathrm{t} \rightarrow[6,5,6]$ |
| slice assignment can <br> remove elements from <br> a list by assigning <br> [] to a slice. |  |  |

## Lists in Environment Diagrams

Assume that before each example below we execute:
s = $[2,3]$
$t=[5,6]$

| Operation | Example | Result |
| :---: | :---: | :---: |
| pop removes \& returns the last element | t = s.pop() | $\begin{aligned} & s \rightarrow[2] \\ & t \rightarrow 3 \end{aligned}$ |
| remove removes the first element equal to the argument | $\begin{aligned} & \text { t.extend(t) } \\ & \text { t. remove(5) } \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3] \\ & t \rightarrow[6,5,6] \end{aligned}$ |
| slice assignment can remove elements from a list by assigning [] to a slice. | $\begin{aligned} & \mathrm{s}[: 1]=[] \\ & \mathrm{t}[0: 2]=[] \end{aligned}$ |  |

## Lists in Environment Diagrams

Assume that before each example below we execute:
s = $[2,3]$
$t=[5,6]$

| Operation | Example | Result |
| :---: | :---: | :---: |
| pop removes \& returns the last element | t = s.pop() | $\begin{aligned} & s \rightarrow[2] \\ & t \rightarrow 3 \end{aligned}$ |
| remove removes the first element equal to the argument | $\begin{aligned} & \text { t.extend(t) } \\ & \text { t. remove(5) } \end{aligned}$ | $\begin{aligned} & s \rightarrow[2,3] \\ & t \rightarrow[6,5,6] \end{aligned}$ |
| slice assignment can remove elements from a list by assigning [] to a slice. | $\begin{aligned} & s[: 1]=[] \\ & t[0: 2]=[] \end{aligned}$ | $\begin{array}{ll} s \rightarrow[3] \\ t \rightarrow[] \end{array}$ |

Lists in Lists in Lists in Environment Diagrams

```
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)
```

$t=[[1,2],[3,4]]$
t[0]. append(t[1:2])

Lists in Lists in Lists in Environment Diagrams

$$
\begin{aligned}
& t=[1,2,3] \\
& t[1: 3]=[t] \\
& \text { t.extend }(\mathrm{t})
\end{aligned}
$$


$t=[[1,2],[3,4]]$
t[0]. append(t[1:2])

Lists in Lists in Lists in Environment Diagrams

$$
\begin{aligned}
& \mathrm{t}=[1,2,3] \\
& \mathrm{t}[1: 3]=[\mathrm{t}] \\
& \mathrm{t} . \text { extend }(\mathrm{t})
\end{aligned}
$$


[t] evaluates to:
$t=[[1,2],[3,4]]$
t[0]. append(t[1:2])

Lists in Lists in Lists in Environment Diagrams

$$
\begin{aligned}
& t=[1,2,3] \\
& t[1: 3]=[t] \\
& \text { t.extend }(\mathrm{t})
\end{aligned}
$$


$t=[[1,2],[3,4]]$
t[0]. append(t[1:2])

Lists in Lists in Lists in Environment Diagrams

$$
\begin{aligned}
& t=[1,2,3] \\
& t[1: 3]=[t] \\
& \text { t.extend }(\mathrm{t})
\end{aligned}
$$


$t=[[1,2],[3,4]]$
t[0]. append(t[1:2])

Lists in Lists in Lists in Environment Diagrams

$$
\begin{aligned}
& t=[1,2,3] \\
& t[1: 3]=[t] \\
& t . \text { extend }(t)
\end{aligned}
$$


$t=[[1,2],[3,4]]$
t[0]. append(t[1:2])

Lists in Lists in Lists in Environment Diagrams

$$
\begin{aligned}
& t=[1,2,3] \\
& t[1: 3]=[t] \\
& \text { t.extend }(\mathrm{t})
\end{aligned}
$$


$t=[[1,2],[3,4]]$
t[0]. append(t[1:2])

Lists in Lists in Lists in Environment Diagrams

[1, [...], 1, [...]]
$t=[[1,2],[3,4]]$
t[0]. append(t[1:2])

## Lists in Lists in Lists in Environment Diagrams

```
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)
```


[1, [...], 1, [...]]
$t=[[1,2],[3,4]]$ t[0]. append(t[1:2])


| list |
| :--- |
| 0 1 |

## Lists in Lists in Lists in Environment Diagrams

```
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)
```


[1, [...], 1, [...]]
$\mathrm{t}=[[1,2],[3,4]]$ t[0]. append(t[1:2])


## Lists in Lists in Lists in Environment Diagrams

```
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)
```


[1, [...], 1, [...]]
$t=[[1,2],[3,4]]$ t[0]. append(t[1:2])


## Lists in Lists in Lists in Environment Diagrams

```
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)
```


[1, [...], 1, [...]]
$t=[[1,2],[3,4]]$ t[0]. append(t[1:2])

[[1, 2, [[3, 4]]], [3, 4]]

