# Mutability

# Announcements

## Objects

(Demo)

# 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

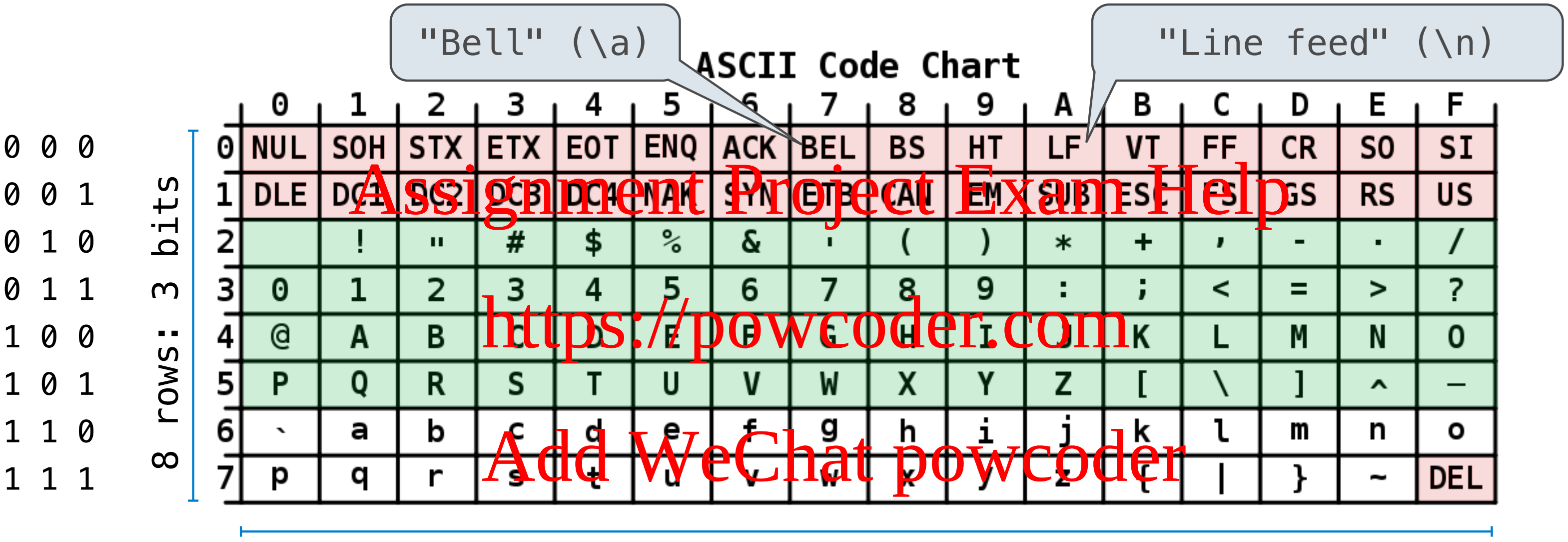
- In Python, every value is an object
  - All **objects** have **attributes**
  - A lot of data manipulation happens through object **methods**
  - Functions do one thing; objects do many related things

# Example: Strings

(Demo)

# Representing Strings: the ASCII Standard

American Standard Code for Information Interchange



"Bell" (\a)

"Line feed" (\n)

**ASCII Code Chart**

8 rows: 3 bits

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

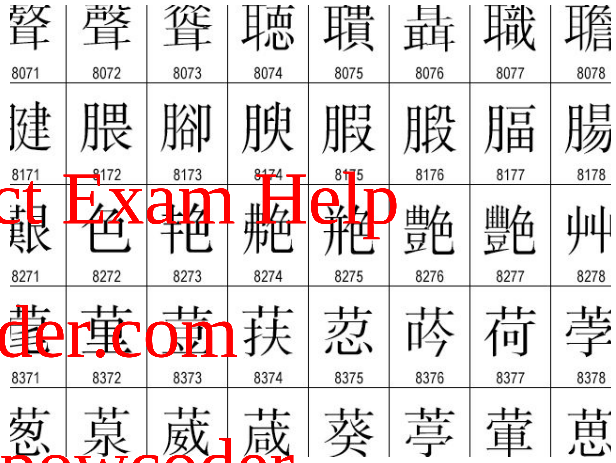
16 columns: 4 bits

- 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

(Demo)

# 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



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 聱<br>8071 | 聲<br>8072 | 聳<br>8073 | 聴<br>8074 | 聵<br>8075 | 聶<br>8076 | 職<br>8077 | 聸<br>8078 |
| 腱<br>8171 | 腲<br>8172 | 腳<br>8173 | 腴<br>8174 | 腵<br>8175 | 腶<br>8176 | 腷<br>8177 | 腸<br>8178 |
| 艱<br>8271 | 色<br>8272 | 艳<br>8273 | 艴<br>8274 | 艵<br>8275 | 艶<br>8276 | 艷<br>8277 | 艸<br>8278 |
| 荱<br>8371 | 荲<br>8372 | 荳<br>8373 | 荴<br>8374 | 荵<br>8375 | 荶<br>8376 | 荷<br>8377 | 荸<br>8378 |
| 葱 | 葲 | 葳 | 葴 | 葵 | 葶 | 葷 | 葸 |

http://ian-albert.com/unicode_chart/unichart-chinese.jpg

LATIN CAPITAL LETTER A

DIE FACE-6 '⚅'

EIGHTH NOTE '♪'

(Demo)

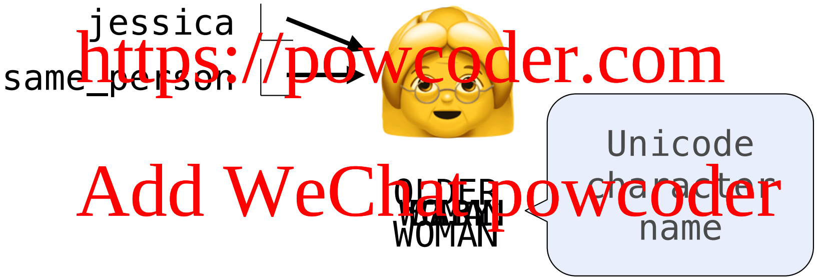
## Mutation Operations

## 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

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

```
def mystery(s):      or   def mystery(s):
    s.pop()                   s[2:] = []
    s.pop()
```

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

```
def another_mystery():
    four.pop()
    four.pop()
```

pythontutor.com/composingprograms.html#code=def%20mystery%28s%29%3A%0A%20%20%20%20s.pop%28%29%0A%20%20%20%20s.pop%28%29%0A%0Afour%20%3D%20[1,%202,%203,%204]%0Amystery%28four%29&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=0

# Tuples

(Demo)

## Tuples are Immutable Sequences

Immutable values are protected from mutation

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

Next lecture: ooze can change turtle's binding

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

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

**Name change:**

```
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

**Object mutation:**

```
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
>>> x.append(3)
>>> x + x
[1, 2, 3, 1, 2, 3]
```

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

- 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
```

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

# 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

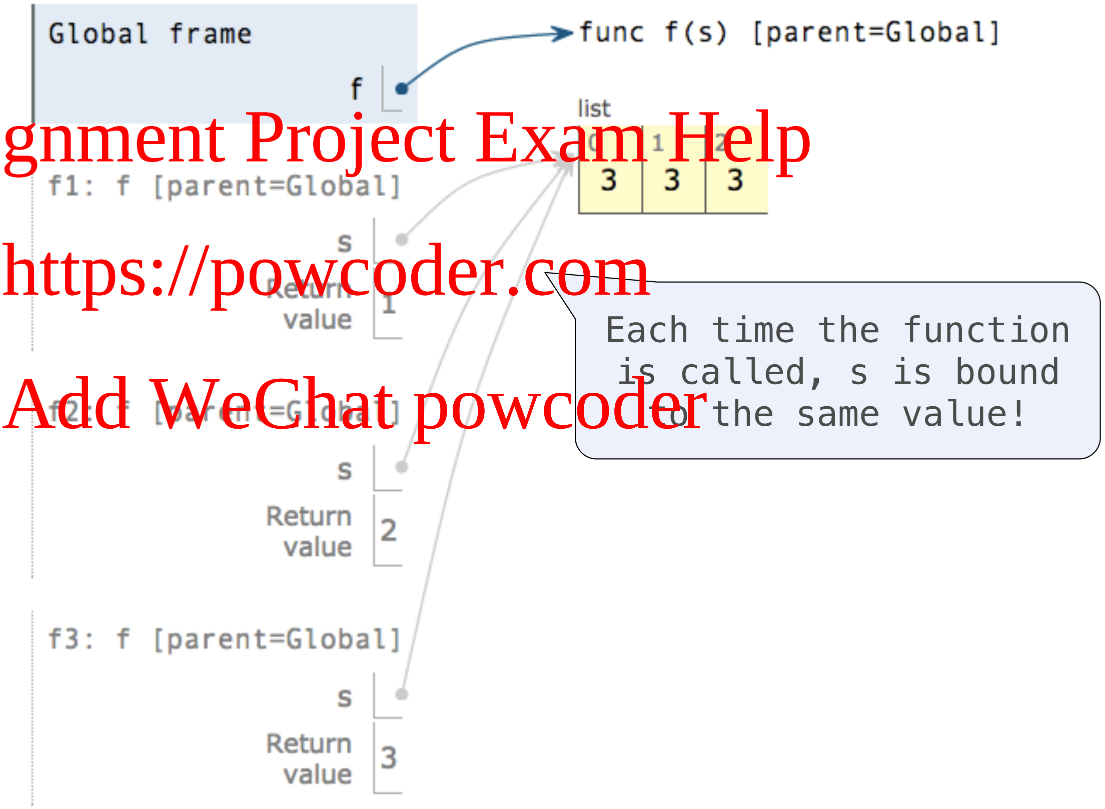
**Identical objects are always equal values**

(Demo)

# Mutable Default Arguments are Dangerous

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
```



Each time the function is called, s is bound to the same value!

pythontutor.com/composingprograms.html#code=def%20f%28s%3D[]%29%3A%0A%20%20%20%20s.append%283%29%0A%20%20%20%20return%20len%28s%29%0A%20%20%20%20%0Af%28%29%0Af%28%29%0Af%28%29&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=0

# Mutable Functions

# A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

```
>>> withdraw = make_withdraw_list(100)

>>> withdraw(25)
75

>>> withdraw(25)
50

>>> withdraw(60)
'Insufficient funds'

>>> withdraw(15)
35
```

In a (mutable) list referenced in the parent frame of the function

Return value: remaining balance

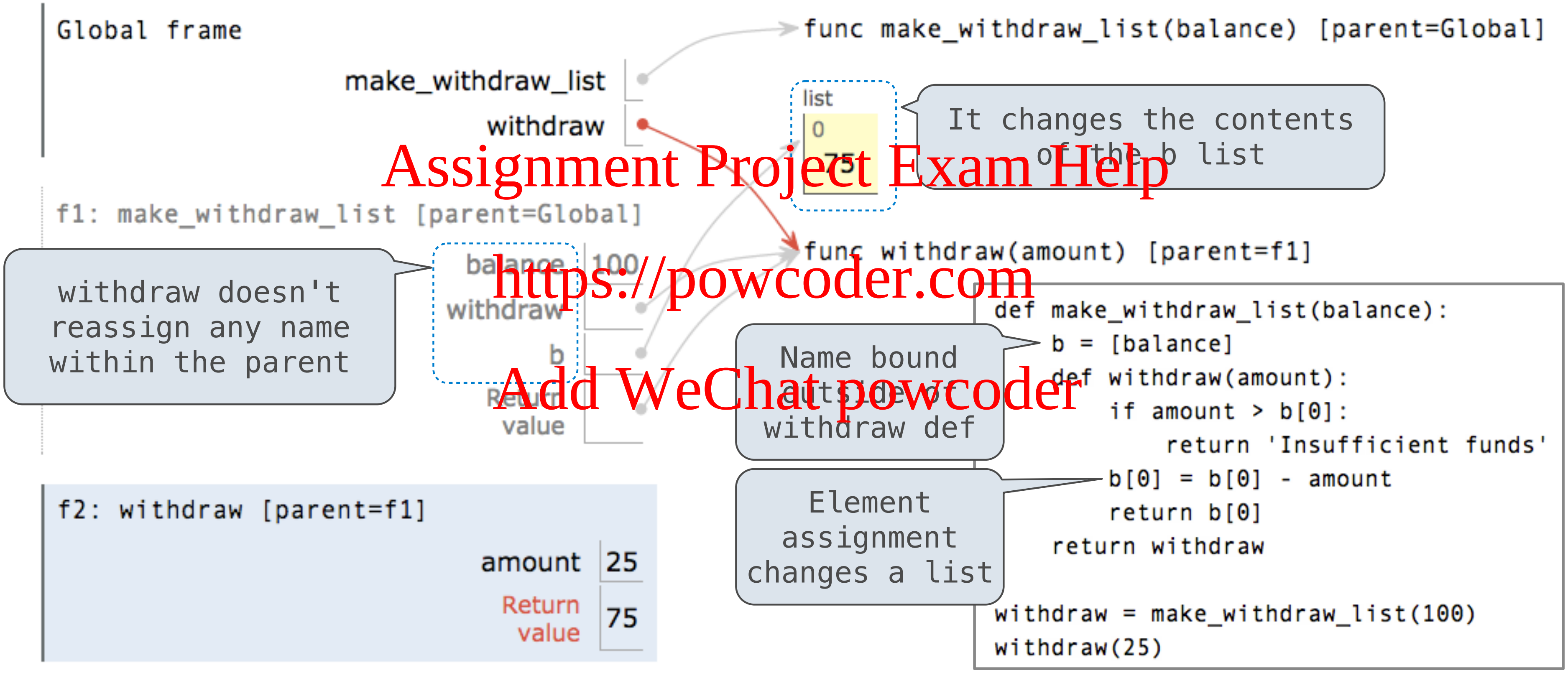
Argument: amount to withdraw

Different return value!

Second withdrawal of the same amount

Where's this balance stored?

# Mutable Values & Persistent Local State



It changes the contents of the b list

withdraw doesn't reassign any name within the parent

Name bound outside of withdraw def

Element assignment changes a list

```
def make_withdraw_list(balance):
    b = [balance]
    def withdraw(amount):
        if amount > b[0]:
            return 'Insufficient funds'
        b[0] = b[0] - amount
        return b[0]
    return withdraw

withdraw = make_withdraw_list(100)
withdraw(25)
```