

# References to objects

**X = 1000**

**X = 1000**



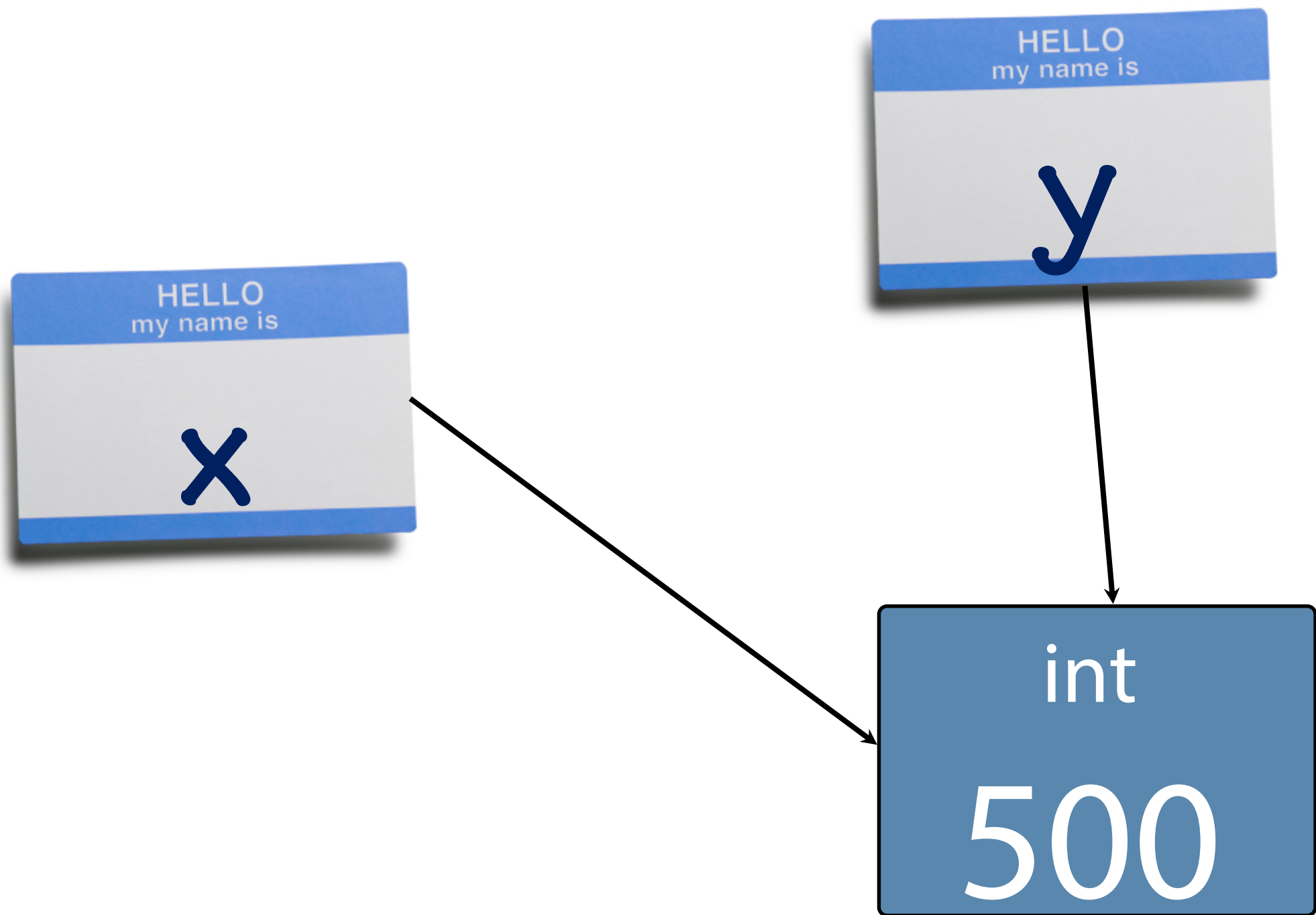
**X = 500**



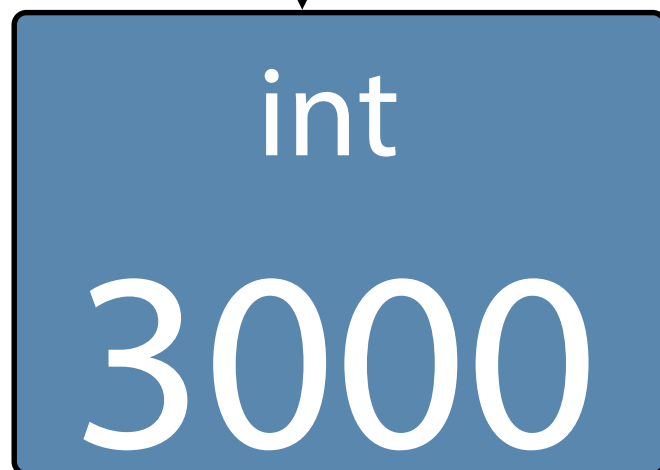
int  
1000

int  
500

$$y = x$$



$x = 3000$

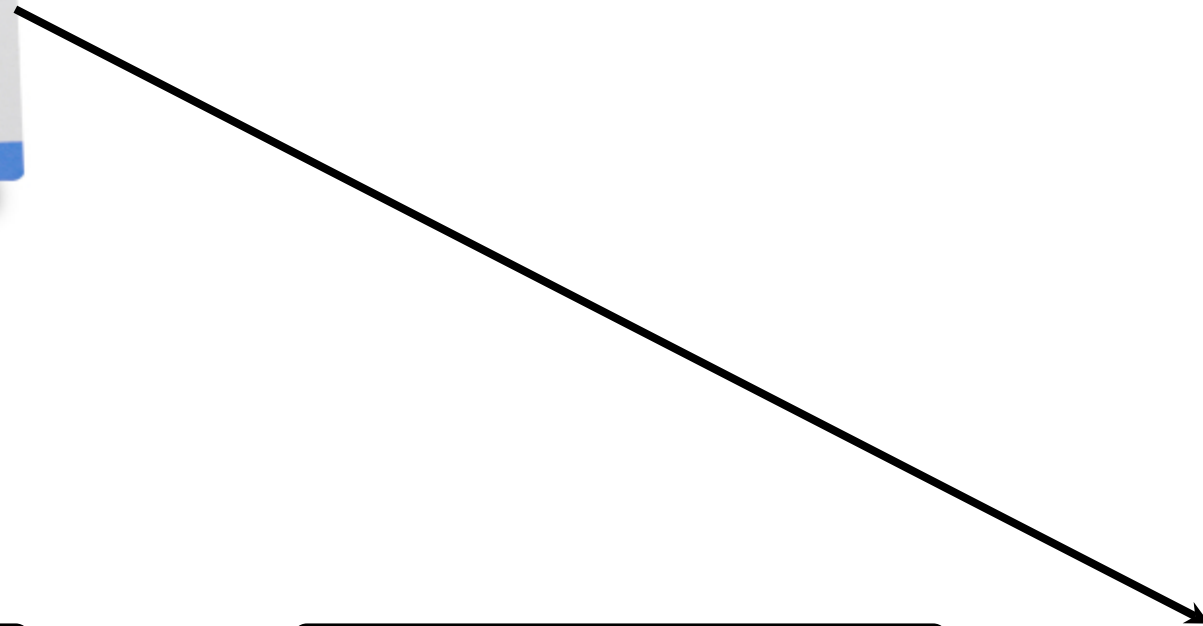




# id()

returns a unique identifier for an object





int  
5

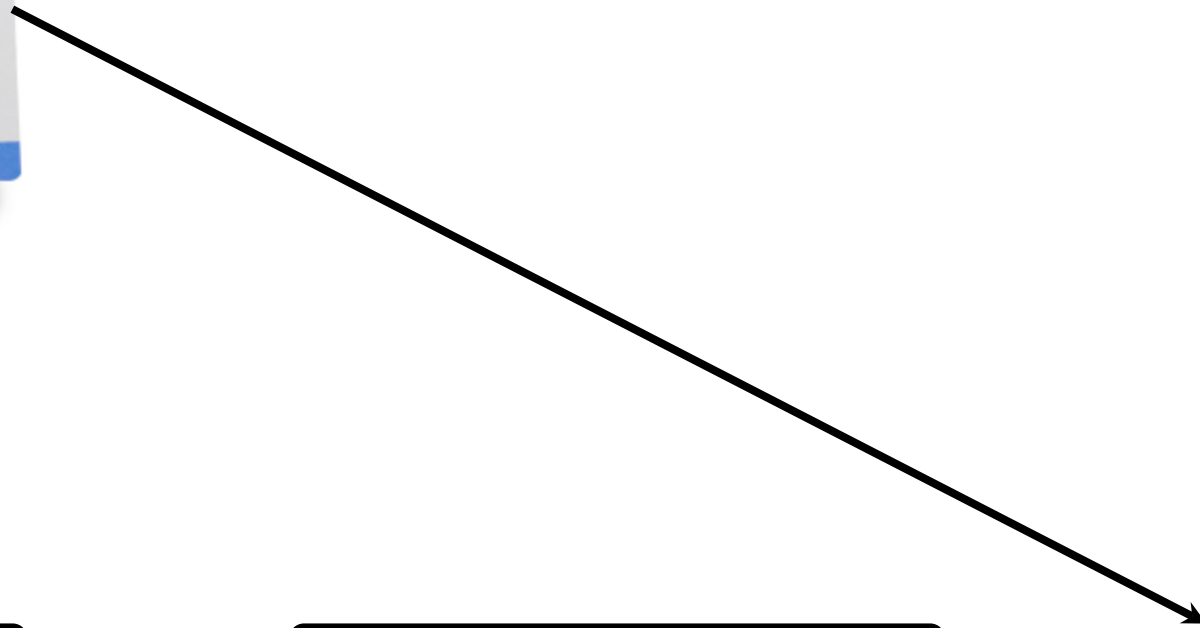
+

int  
2

=

int  
7



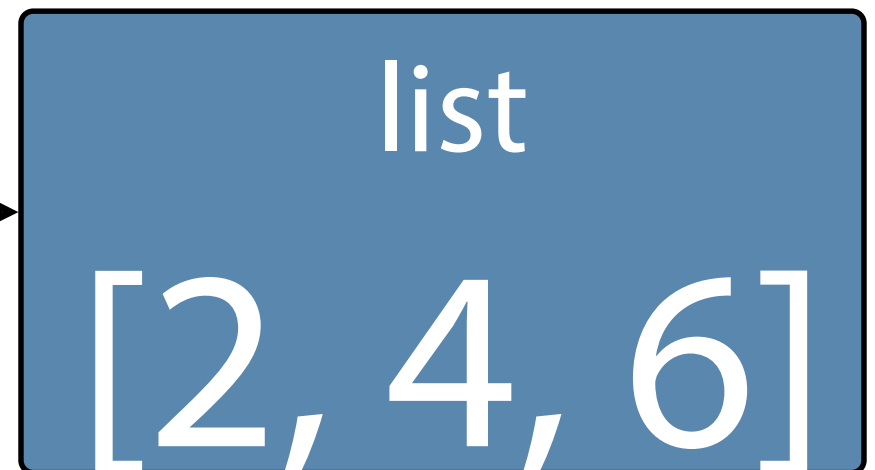


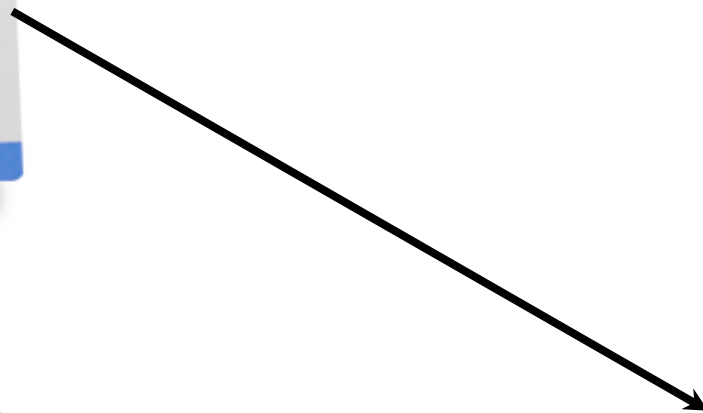
int  
5

int  
2

int  
7







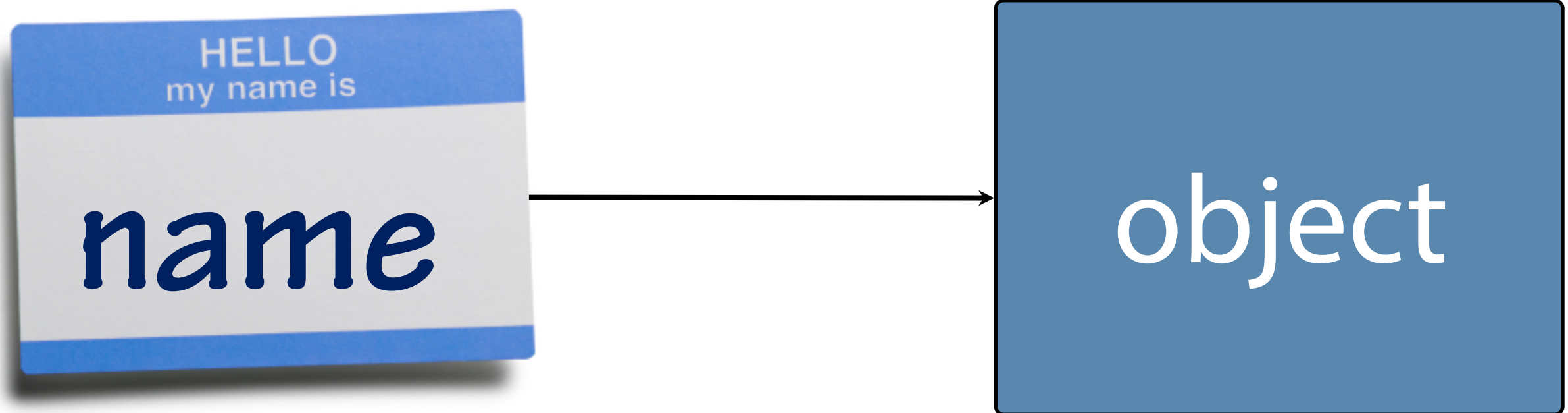
id() deals with the object, not  
the reference

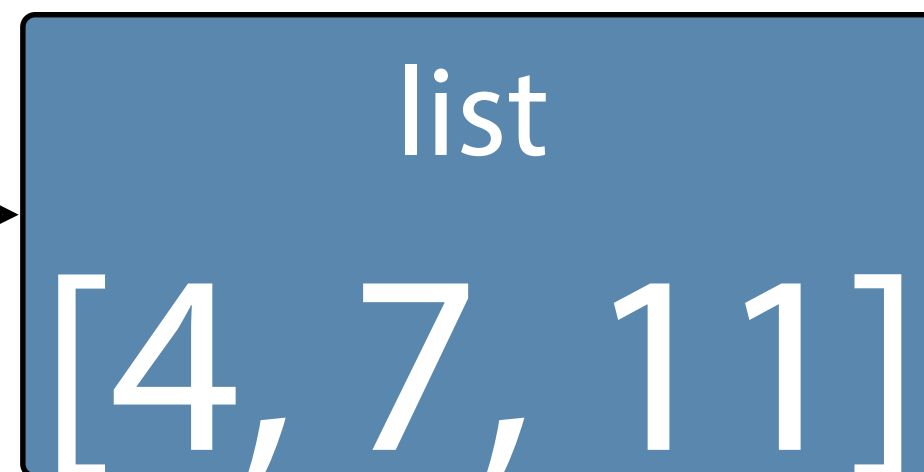
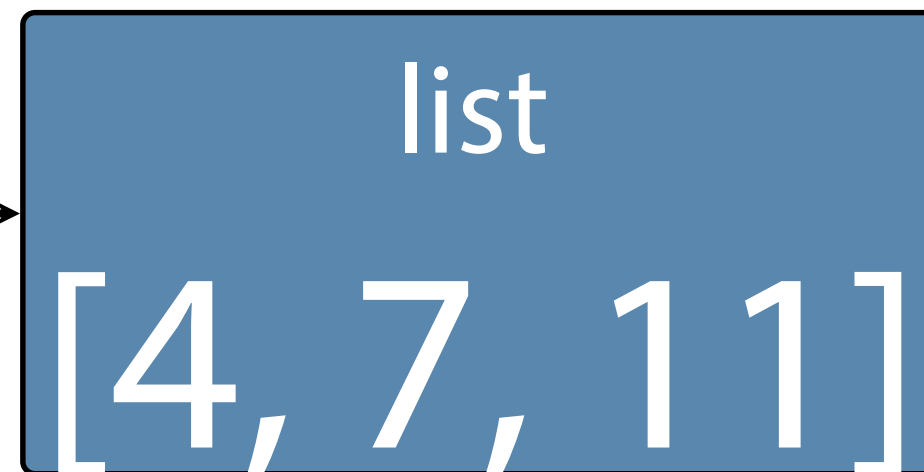


list  
[2, 4, 6]

# ~~Variables~~

Named references to objects



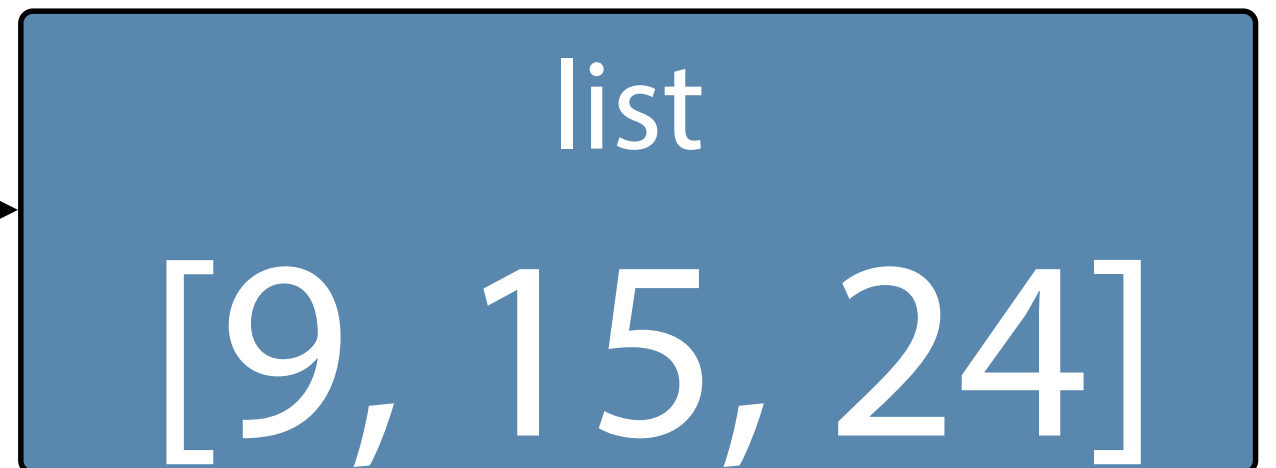


# Value equality vs. identity

- **Value** - equivalent “contents”  
**Identity** - same object

Value comparison can be controlled programmatically.





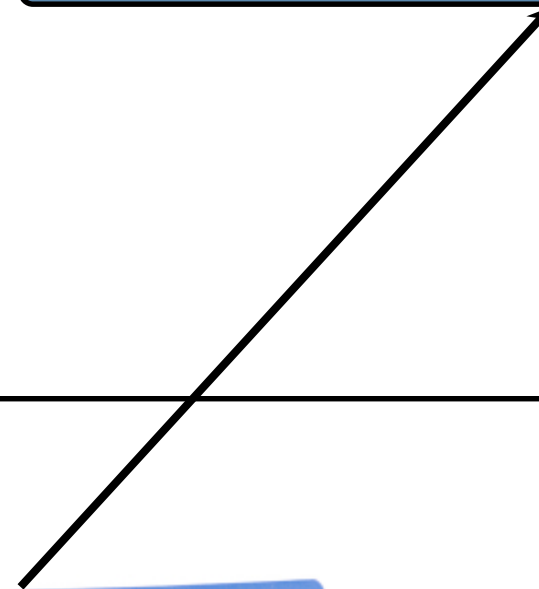


modify(k):



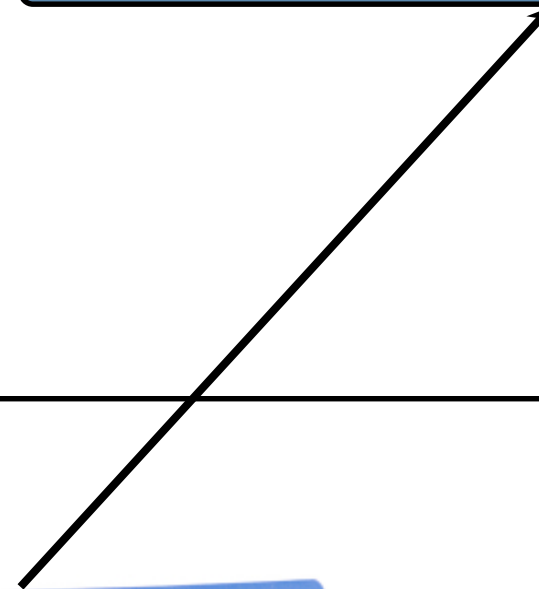


modify(**k**):



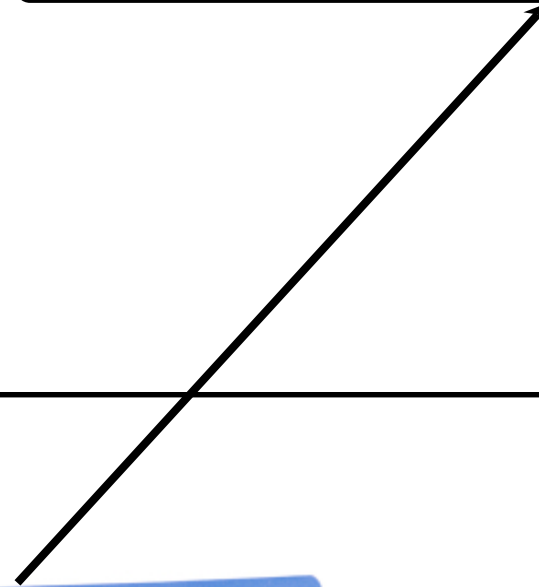


modify(k):

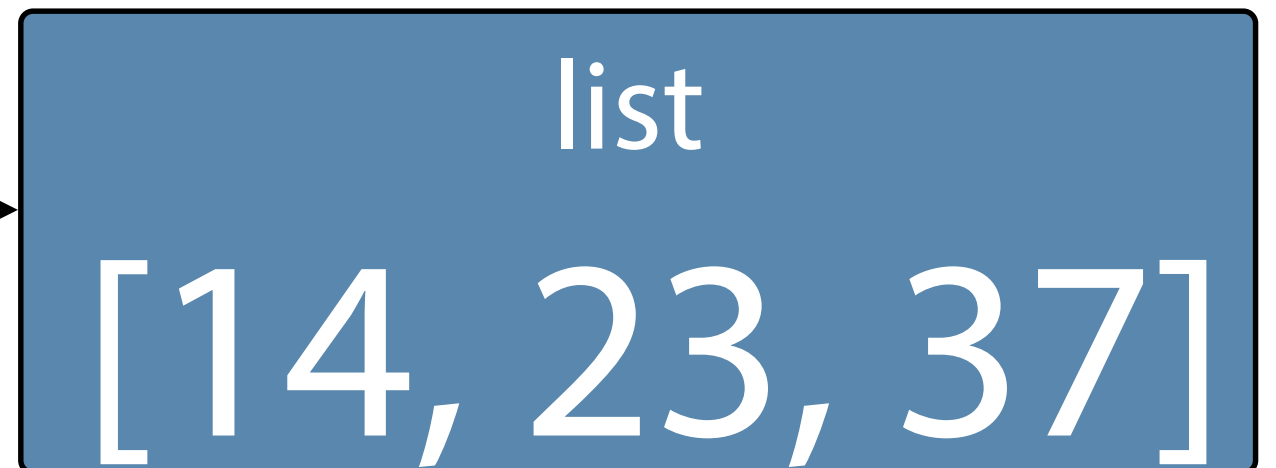




modify(k):

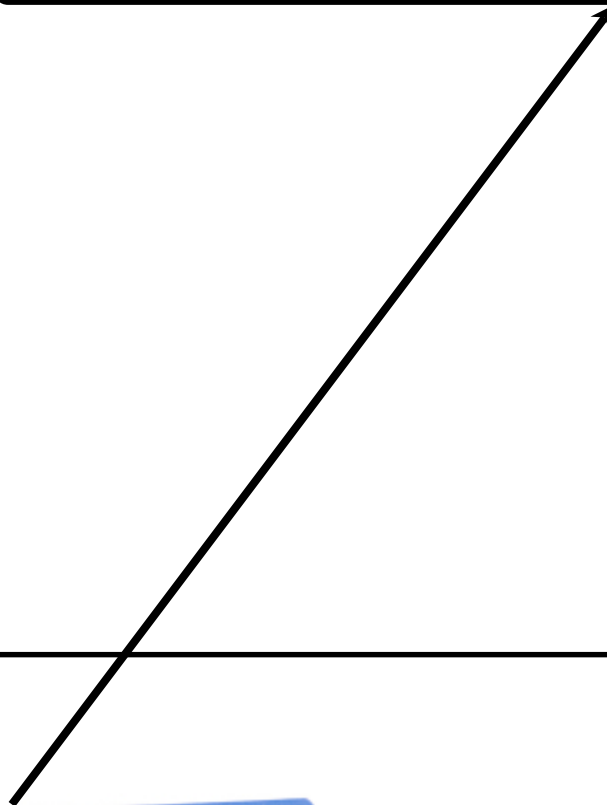




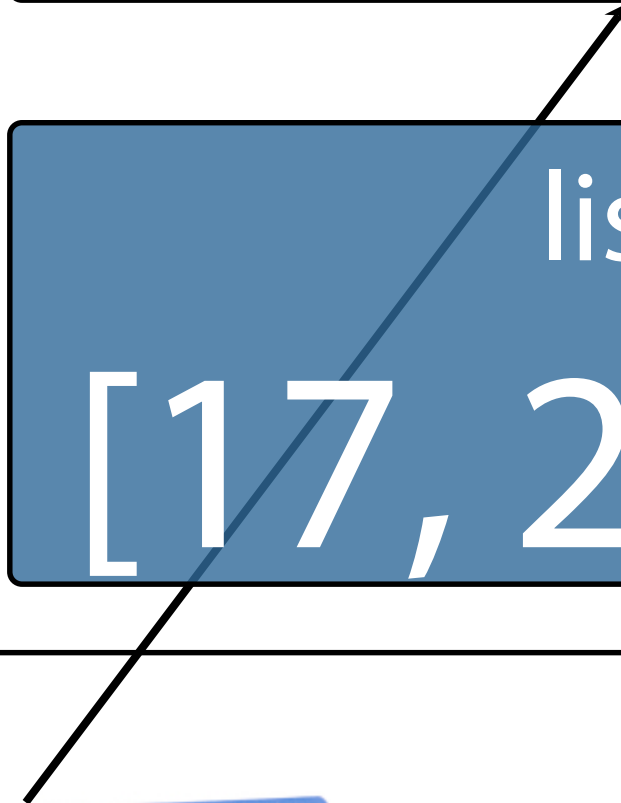




replace(g):







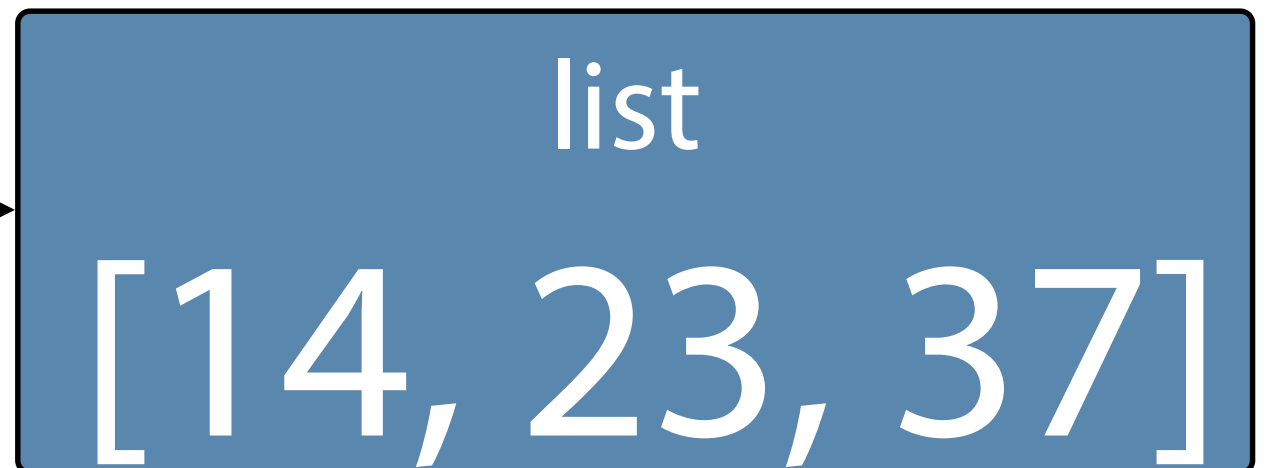
replace(g):



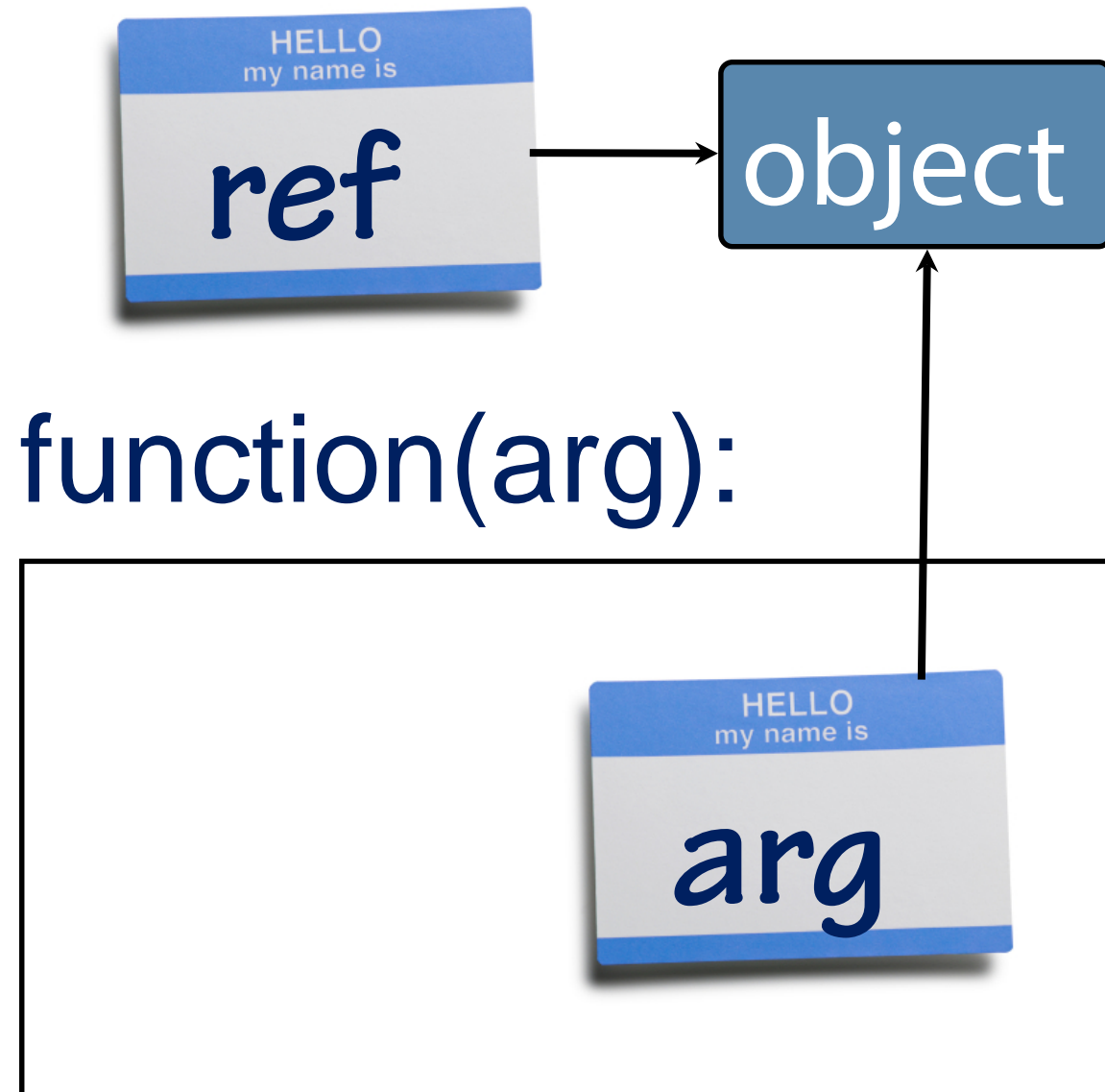


replace(g):





# Pass By Object Reference



- The value of the **reference** is copied, not the value of the **object**.

# Default Arguments

```
def function(a, b=value)
```

Default value for 'b'



# Default Argument Evaluation

- Default argument values are evaluated when def is evaluated.
- They can be modified like any other object.

```
def add_spam(menu=
```

```
):  
    list
```

```
    []
```



```
def add_spam(menu=
```

```
):  
    list
```

```
    ["spam"]
```





```
def add_spam(menu=
```

```
):  
    list
```

```
    ["spam",  
     "spam"]
```



```
def add_spam(menu=
```

```
):  
    list
```

```
    ["spam",  
     "spam",  
     "spam"]
```



Static

Dynamic

Strong



Haskell

C++



Weak



Static

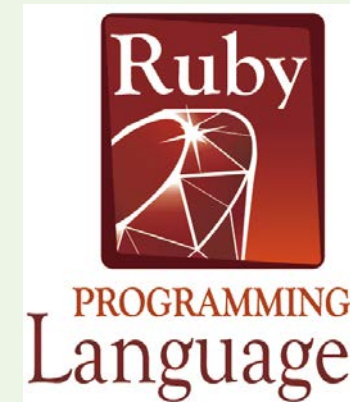
Dynamic

Strong



Haskell

C++



Weak

JS



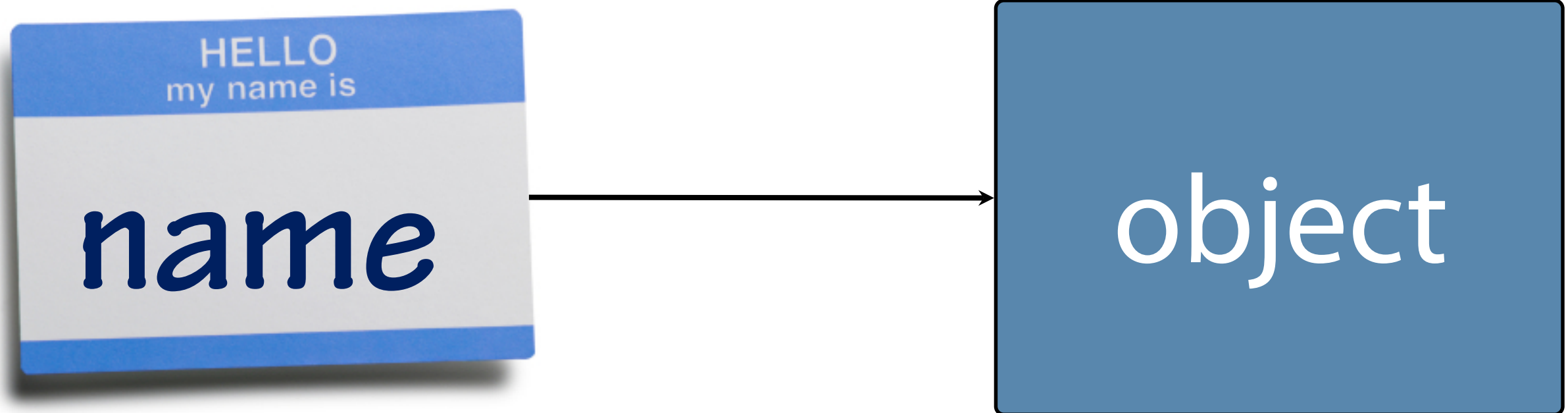
# Dynamic Type System

- In a dynamic type system object types are only resolved at runtime.

# Strong Type System

- In a strong type system there is no implicit type conversion.

# Object References Have No Type



# Python Name Scopes

Scopes are contexts in which  
named references can be  
looked up.



# Python Name Scopes

Local

Inside the current function

# Python Name Scopes

**Local**      Inside the current function

**Enclosing**      Any and all enclosing functions

# Python Name Scopes

Local	Inside the current function
Enclosing	Any and all enclosing functions
Global	Top-level of module

# Python Name Scopes

Local	Inside the current function
Enclosing	Any and all enclosing functions
Global	Top-level of module
Built-in	Provided by the builtins module

# Python Name Scopes

Local

Enclosing

Global

Built-in

```
#!/usr/bin/env python3
"""Retrieve and print words from a URL.
```

Tools to read a UTF-8 text document from a URL which will be split into its component words for printing.

Script usage:

```
python3 words.py <URL>
"""
```

```
import sys
from urllib.request import urlopen
```

```
def fetch_words(url):
    """Fetch a list of words from a URL.
```

```
    Args:
        url: The URL of a UTF-8 text document.
```

```
    Returns:
        A list of strings containing the words from
        the document.
    """
```

```
    with urlopen(url) as story:
        story_words = []
        for line in story:
            line_words = line.decode('utf8').split()
            for word in line_words:
                story_words.append(word)
    print(locals())
    return story_words
```

```
def print_items(items):
    """Print items one per line.
```

```
    Args:
        An iterable series of printable items.
    """
```

```
    for item in items:
        print(item)
```

```
def main(url):
    """Print each word from a text document from at a URL.
```

```
    Args:
        url: The URL of a UTF-8 text document.
    """
    words = fetch_words(url)
    print_items(words)
```

```
if __name__ == '__main__':
    main(sys.argv[1]) # The 0th arg is the module filename.
```

```
#!/usr/bin/env python3
"""Retrieve and print words from a URL.
```

Tools to read a UTF-8 text document from a URL which will be split into its component words for printing.

Script usage:

```
python3 words.py <URL>
"""
```

```
import sys
from urllib.request import urlopen
```

```
def fetch_words(url):
    """Fetch a list of words from a URL.
```

```
    Args:
        url: The URL of a UTF-8 text document.
```

```
    Returns:
        A list of strings containing the words from
        the document.
    """
```

```
    with urlopen(url) as story:
        story_words = []
        for line in story:
            line_words = line.decode('utf8').split()
            for word in line_words:
                story_words.append(word)
    print(locals())
    return story_words
```

```
def print_items(items):
    """Print items one per line.
```

```
    Args:
        An iterable series of printable items.
    """
```

```
    for item in items:
        print(item)
```

```
def main(url):
```

```
    """Print each word from a text document from at a URL.
```

```
    Args:
        url: The URL of a UTF-8 text document.
    """
```

```
    words = fetch_words(url)
    print_items(words)
```

```
if __name__ == '__main__':
    main(sys.argv[1]) # The 0th arg is the module filename.
```

```
#!/usr/bin/env python3
"""Retrieve and print words from a URL.
```

Tools to read a UTF-8 text document from a URL which will be split into its component words for printing.

Script usage:

```
python3 words.py <URL>
"""
```

```
import sys
from urllib.request import urlopen
```

```
def fetch_words(url):
    """Fetch a list of words from a URL.
```

Args:  
url: The URL of a UTF-8 text document.

Returns:  
A list of strings containing the words from the document.

```
"""
with urlopen(url) as story:
    story_words = []
    for line in story:
        line_words = line.decode('utf8').split()
        for word in line_words:
            story_words.append(word)
print(locals())
return story_words
```

```
def print_items(items):
    """Print items one per line.
```

Args:  
An iterable series of printable items.

```
"""
for item in items:
    print(item)
```

```
def main(url):
    """Print each word from a text document from at a URL.

    Args:
        url: The URL of a UTF-8 text document.
    """
    words = fetch_words(url)
    print_items(words)

if __name__ == '__main__':
    main(sys.argv[1]) # The 0th arg is the module filename.
```



```
#!/usr/bin/env python3
"""Retrieve and print words from a URL.
```

Tools to read a UTF-8 text document from a URL which will be split into its component words for printing.

Script usage:

```
python3 words.py <URL>
"""
```

```
import sys
from urllib.request import urlopen
```

```
def fetch_words(url):
    """Fetch a list of words from a URL.
```

Args:  
url: The URL of a UTF-8 text document.

Returns:  
A list of strings containing the words from the document.

```
"""
with urlopen(url) as story:
    story_words = []
    for line in story:
        line_words = line.decode('utf8').split()
        for word in line_words:
            story_words.append(word)
print(locals())
return story_words
```

```
def print_items(items):
    """Print items one per line.
```

Args:  
An iterable series of printable items.

```
"""
for item in items:
    print(item)
```

```
def main(url):
    """Print each word from a text document from at a URL.
```

Args:  
url: The URL of a UTF-8 text document.

```
"""
words = fetch_words(url)
print_items(words)
```

```
if __name__ == '__main__':
    main(sys.argv[1]) # The 0th arg is the module filename.
```

```
#!/usr/bin/env python3
"""Retrieve and print words from a URL.
```

Tools to read a UTF-8 text document from a URL which will be split into its component words for printing.

Script usage:

```
python3 words.py <URL>
"""
```

```
import sys
```

```
from urllib.request import urlopen
```

```
def fetch_words(url):
    """Fetch a list of words from a URL.
```

Args:  
url: The URL of a UTF-8 text document.

Returns:  
A list of strings containing the words from the document.  
"""

```
with urlopen(url) as story:
    story_words = []
    for line in story:
        line_words = line.decode('utf8').split()
        for word in line_words:
            story_words.append(word)
    print(locals())
    return story_words
```

```
def print_items(items):
    """Print items one per line.
```

Args:  
An iterable series of printable items.  
"""

```
for item in items:
    print(item)
```

```
def main(url):
```

"""Print each word from a text document from at a URL.

Args:  
url: The URL of a UTF-8 text document.  
"""

```
words = fetch_words(url)
print_items(words)
```

```
if __name__ == '__main__':
```

```
    main(sys.argv[1]) # The 0th arg is the module filename.
```

```
#!/usr/bin/env python3
"""Retrieve and print words from a URL.
```

Tools to read a UTF-8 text document from a URL which will be split into its component words for printing.

Script usage:

```
python3 words.py <URL>
"""
```

```
import sys
```

```
from urllib.request import urlopen
```

```
def fetch_words(url):
```

```
    """Fetch a list of words from a URL.
```

Args:

url: The URL of a UTF-8 text document.

Returns:

A list of strings containing the words from the document.

```
    """
```

```
    with urlopen(url) as story:
```

```
        story_words = []
```

```
        for line in story:
```

```
            line_words = line.decode('utf8').split()
```

```
            for word in line_words:
```

```
                story_words.append(word)
```

```
    print(locals())
```

```
    return story_words
```

```
def print_items(items):
```

```
    """Print items one per line.
```

Args:

An iterable series of printable items.

```
    """
```

```
    for item in items:
```

```
def main(url):
```

```
    """Print each word from a text document from at a URL.
```

Args:

url: The URL of a UTF-8 text document.

```
    """
```

```
    words = fetch_words(url)
```

```
    print_items(words)
```

```
if __name__ == '__main__':
```

```
    main(sys.argv[1]) # The 0th arg is the module filename.
```

```
#!/usr/bin/env python3
"""Retrieve and print words from a URL.
```

Tools to read a UTF-8 text document from a URL which will be split into its component words for printing.

Script usage:

```
python3 words.py <URL>
"""
```

```
import sys
```

```
from urllib.request import urlopen
```

```
def fetch_words(url):
```

```
    """Fetch a list of words from a URL.
```

Args:

url: The URL of a UTF-8 text document.

Returns:

A list of strings containing the words from the document.

```
    """
```

```
    with urlopen(url) as story:
```

```
        story_words = []
```

```
        for line in story:
```

```
            line_words = line.decode('utf8').split()
```

```
            for word in line_words:
```

```
                story_words.append(word)
```

```
    print(locals())
```

```
    return story_words
```

```
def print_items(items):
```

```
    """Print items one per line.
```

Args:

An iterable series of printable items.

```
def main(url):
```

```
    """Print each word from a text document from at a URL.
```

Args:

url: The URL of a UTF-8 text document.

```
    """
```

```
    words = fetch_words(url)
```

```
    print_items(words)
```

```
if __name__ == '__main__':
```

```
    main(sys.argv[1]) # The 0th arg is the module filename.
```

```
def fetch_words(url):  
    """Fetch a list of words from a URL.
```

Args:

url: The URL of a UTF-8 text document.

Returns:

A list of strings containing the words from  
the document.

```
    """
```

```
    with urlopen(url) as story:  
        story_words = []  
        for line in story:  
            line_words = line.decode('utf8').split()  
            for word in line_words:  
                story_words.append(word)  
    print(locals())  
    return story_words
```

```
def fetch_words(url):  
    """Fetch a list of words from a URL.
```

Args:

url: The URL of a UTF-8 text document.

Returns:

A list of strings containing the words from  
the document.

```
    """
```

```
    with urlopen(url) as story:
```

```
        story_words = []
```

```
        for line in story:
```

```
            line_words = line.decode('utf8').split()
```

```
            for word in line_words:
```

```
                story_words.append(word)
```

```
    print(locals())
```

```
    return story_words
```

```
def fetch_words(url):  
    """Fetch a list of words from a URL.
```

Args:

url: The URL of a UTF-8 text document.

Returns:

A list of strings containing the words from  
the document.

```
    """
```

```
    with urlopen(url) as story:
```

```
        story_words = []
```

```
        for line in story:
```

```
            line_words = line.decode('utf8').split()
```

```
            for word in line_words:
```

```
                story_words.append(word)
```

```
    print(locals())
```

```
    return story_words
```

```
def fetch_words(url):  
    """Fetch a list of words from a URL.  
  
    Args:  
        url: The URL of a UTF-8 text document.  
  
    Returns:  
        A list of strings containing the words from  
        the document.  
    """  
    with urlopen(url) as story:  
        story_words = []  
        for line in story:  
            line_words = line.decode('utf8').split()  
            for word in line_words:  
                story_words.append(word)  
    print(locals())  
    return story_words
```



```
def fetch_words(url):  
    """Fetch a list of words from a URL.
```

Args:

url: The URL of a UTF-8 text document.

Returns:

A list of strings containing the words from  
the document.

```
    """
```

```
    with urlopen(url) as story:
```

```
        story_words = []
```

```
        for line in story:
```

```
            line_words = line.decode('utf8').split()
```

```
            for word in line_words:
```

```
                story_words.append(word)
```

```
    print(locals())
```

```
    return story_words
```

```
def fetch_words(url):
    """Fetch a list of words from a URL.

    Args:
        url: The URL of a UTF-8 text document.

    Returns:
        A list of strings containing the words from
        the document.
    """
    with urlopen(url) as story:
        story_words = []
        for line in story:
            line_words = line.decode('utf8').split()
            for word in line_words:
                story_words.append(word)
        print(locals())
    return story_words
```

```
def fetch_words(url):
    """Fetch a list of words from a URL.

    Args:
        url: The URL of a UTF-8 text document.

    Returns:
        A list of strings containing the words from
        the document.
    """

    with urlopen(url) as story:
        story_words = []
        for line in story:
            line_words = line.decode('utf8').split()
            for word in line_words:
                story_words.append(word)
    print(locals())
    return story_words
```



# global

rebinds a global name at module scope

```
"""Demonstrate scoping."""
```

```
count = 0
```

```
def show_count():  
    print("count = ", count)
```

```
def set_count(c):  
    count = c
```

```
"""Demonstrate scoping."""
```

```
count = 0
```

```
def show_count():  
    print("count = ", count)
```

```
def set_count(c):  
    global count  
    count = c
```

# Everything is an object





# Objects – Summary





# Objects – Summary

- Think of named references to objects rather than variables
  - Assignment attaches a name to an object
  - Assigning from one reference to another puts two name tags on the same object.



# Objects – Summary

- Think of named references to objects rather than variables
  - Assignment attaches a name to an object
  - Assigning from one reference to another puts two name tags on the same object.
- The garbage collector reclaims unreachable objects



# Objects – Summary

- Think of named references to objects rather than variables
  - Assignment attaches a name to an object
  - Assigning from one reference to another puts two name tags on the same object.
- The garbage collector reclaims unreachable objects
- `id()` returns a unique and constant identifier
  - rarely used in production



# Objects – Summary

- Think of named references to objects rather than variables
  - Assignment attaches a name to an object
  - Assigning from one reference to another puts two name tags on the same object.
- The garbage collector reclaims unreachable objects
- `id()` returns a unique and constant identifier
  - rarely used in production
- The `is` operator determines equality of identity



# Objects – Summary

- Think of named references to objects rather than variables
  - Assignment attaches a name to an object
  - Assigning from one reference to another puts two name tags on the same object.
- The garbage collector reclaims unreachable objects
- `id()` returns a unique and constant identifier
  - rarely used in production
- The `is` operator determines equality of identity
- Test for equivalence using `==`



# Objects – Summary

- Think of named references to objects rather than variables
  - Assignment attaches a name to an object
  - Assigning from one reference to another puts two name tags on the same object.
- The garbage collector reclaims unreachable objects
- `id()` returns a unique and constant identifier
  - rarely used in production
- The `is` operator determines equality of identity
- Test for equivalence using `==`
- Function arguments are passed by object-reference
  - functions can modify mutable arguments



# Objects – Summary

- Think of named references to objects rather than variables
  - Assignment attaches a name to an object
  - Assigning from one reference to another puts two name tags on the same object.
- The garbage collector reclaims unreachable objects
- `id()` returns a unique and constant identifier
  - rarely used in production
- The `is` operator determines equality of identity
- Test for equivalence using `==`
- Function arguments are passed by object-reference
  - functions can modify mutable arguments
- Reference is lost if a formal function argument is rebound
  - To change a mutable argument, replace its contents



# Objects – Summary

- Think of named references to objects rather than variables
  - Assignment attaches a name to an object
  - Assigning from one reference to another puts two name tags on the same object.
- The garbage collector reclaims unreachable objects
- `id()` returns a unique and constant identifier
  - rarely used in production
- The `is` operator determines equality of identity
- Test for equivalence using `==`
- Function arguments are passed by object-reference
  - functions can modify mutable arguments
- Reference is lost if a formal function argument is rebound
  - To change a mutable argument, replace its contents
- `return` also passes by object-reference





# Objects – Summary

- Function arguments can be specified with defaults



# Objects – Summary

- Function arguments can be specified with defaults
- Default argument expressions evaluated once, when def is executed



# Objects – Summary

- Function arguments can be specified with defaults
- Default argument expressions evaluated once, when def is executed
- Python uses dynamic typing
  - We don't specify types in advance



# Objects – Summary

- Function arguments can be specified with defaults
- Default argument expressions evaluated once, when def is executed
- Python uses dynamic typing
  - We don't specify types in advance
- Python uses strong typing
  - Types are not coerced to match



# Objects – Summary

- Function arguments can be specified with defaults
- Default argument expressions evaluated once, when def is executed
- Python uses dynamic typing
  - We don't specify types in advance
- Python uses strong typing
  - Types are not coerced to match
- Names are looked up in four nested scopes
  - LEGB rule: Local, Enclosing, Global, and Built-ins



# Objects – Summary

- Function arguments can be specified with defaults
- Default argument expressions evaluated once, when def is executed
- Python uses dynamic typing
  - We don't specify types in advance
- Python uses strong typing
  - Types are not coerced to match
- Names are looked up in four nested scopes
  - LEGB rule: Local, Enclosing, Global, and Built-ins
- Global references can be read from a local scope



# Objects – Summary

- Function arguments can be specified with defaults
- Default argument expressions evaluated once, when def is executed
- Python uses dynamic typing
  - We don't specify types in advance
- Python uses strong typing
  - Types are not coerced to match
- Names are looked up in four nested scopes
  - LEGB rule: Local, Enclosing, Global, and Built-ins
- Global references can be read from a local scope
- Use global to assign to global references from a local scope



# Objects – Summary

- Function arguments can be specified with defaults
- Default argument expressions evaluated once, when def is executed
- Python uses dynamic typing
  - We don't specify types in advance
- Python uses strong typing
  - Types are not coerced to match
- Names are looked up in four nested scopes
  - LEGB rule: Local, Enclosing, Global, and Built-ins
- Global references can be read from a local scope
- Use global to assign to global references from a local scope
- Everything in Python is an object
  - This includes modules and functions
  - They can be treated just like other objects





# Objects – Summary

import and def result in binding to named references



# Objects – Summary

import and def result in binding to named reference  
type can be used to determine the type of an object



# Objects – Summary

import and def result in binding to named reference  
type can be used to determine the type of an object  
dir() can be used to introspect an object and get its attributes



# Objects – Summary

import and def result in binding to named reference  
type can be used to determine the type of an object  
dir() can be used to introspect an object and get its attributes  
The name of a function or module object can be accessed  
through its `__name__` attribute



# Objects – Summary

- `import` and `def` result in binding to named reference type  
`type()` can be used to determine the type of an object  
`dir()` can be used to introspect an object and get its attributes  
The name of a function or module object can be accessed through its `__name__` attribute  
The docstring for a function or module object can be accessed through its `__doc__` attribute



# Objects – Summary

- `import` and `def` result in binding to named reference type  
can be used to determine the type of an object  
`dir()` can be used to introspect an object and get its attributes  
The name of a function or module object can be accessed through its `__name__` attribute  
The docstring for a function or module object can be accessed through its `__doc__` attribute

Use `len()` to measure the length of a string



# Objects – Summary

- import and def result in binding to named referencestype  
can be used to determine the type of an objectdir() can  
be used to introspect an object and get its attributesThe  
name of a function or module object can be accessed  
through it's `__name__` attributeThe docstring for a  
function or module object can be accessed through its  
`__doc__` attribute
- Use len() to measure the length of a stringYou can  
multiple a string by an integer
  - Produces a new string with multiple copies of the operand
  - This is called the "repetition" operation