Object Examples

Announcements

Composition

Attribute Composition

Attribute values are themselves objects, which have attributes and methods

Any type of object can be an attribute value of any other type

When reading/writing a complicated expression, think about the type of each sub-expression:

- **self** always has the type of the class being defined
- For **self.x.y: self.x** is an object (an attribute of **self**) with an attribute **y.**
- For **self.f(x).y: self.f(x)** is the return value of method **f,** an object with an attribute **y.**
- For **self.f(x).g(y):** the return value of **self**'s method **f** is an object that has a method **g.**

Example: Message Delivery

```
@dataclass
class Email:
    msg: str
    sender: str
    recipient_name: str

class Server:
    def __init__(self):
        self.clients: dict[str, Client] = {}

    def send(self, email: Email) -> None:
        "Append the email to the inbox of the recipient's client."
        ____________________________________________________________

class Client:
    def __init__(self, server: Server, name: str):
        self.server = server
        self.name = name
        self.inbox: list[Email] = []
        server._____________________

    def compose(self, message: str, recipient: str) -> None:
        "Send an email with the given message to the recipient."
        self.server.send(_________________________________________)
```

A **Client** can **compose** an **Email** to **send** it to the **Server**, which appends it to the **inbox** of the **Client** named as the **recipient.**

Each **Server** has a **clients** dictionary from the **name** of each **Client** to that **Client**, and each **Client** adds itself to **clients.**

```
>>> s = Server()
>>> a = Client(s, 'Alice')
>>> b = Client(s, 'Bob')
>>> a.compose('Hello!', 'Bob')
>>> len(b.inbox)
1
>>> b.inbox[0].msg
'Hello!'
```

Example: Message Delivery

```
@dataclass
class Email:
    msg: str
    sender: str
    recipient_name: str

class Server:
    def __init__(self):
        self.clients: dict[str, Client] = {}
    def send(self, email: Email) -> None:
        "Append the email to the inbox of the recipient's client."
        self.clients[email.recipient_name].inbox.append(email)

class Client:
    def __init__(self, server: Server, name: str):
        self.server = server
        self.name = name
        self.inbox: list[Email] = []
        server.clients[name] = self

    def compose(self, message: str, recipient: str) -> None:
        "Send an email with the given message to the recipient."
        self.server.send(Email(message, self.name, recipient))
```

Server dict Client list[Email] Email

A **Client** can **compose** an **Email** to **send** it to the **Server**, which appends it to the **inbox** of the **Client** named as the **recipient.**

Each **Server** has a **clients** dictionary from the **name** of each **Client** to that **Client**, and each **Client** adds itself to **clients.**

```
>>> s = Server()
>>> a = Client(s, 'Alice')
>>> b = Client(s, 'Bob')
>>> a.compose('Hello!', 'Bob')
>>> len(b.inbox)
1
>>> b.inbox[0].msg
'Hello!'
```

Linked List Practice

Building Linked Lists Recursively

```
type LinkedList[T] = Link[T] | tuple[()]

@dataclass
class Link[T]:
    first: T
    rest: LinkedList[T] = ()  # rest defaults to empty
```

Building a linked list using recursion often uses the recursive result as the rest

```
def some_function(...) -> LinkedList:
    ... Link(..., some_function(...)) ...
```

```
def squares(n) -> LinkedList[int]:
    """A linked list of n perfect squares.

    >>> print(squares(10))
    (1 4 9 16 25 36 49 64 81 100)
    """
    def f(i):
        if i > n:
            return ()
        else:
            return ____________________
    return f(1)
```

Building Linked Lists Recursively

```
type LinkedList[T] = Link[T] | tuple[()]

@dataclass
class Link[T]:
    first: T
    rest: LinkedList[T] = ()  # rest defaults to empty
```

Building a linked list using recursion often uses the recursive result as the rest

```
def some_function(...) -> LinkedList:
    ... Link(..., some_function(...)) ...
```

```
def squares(n) -> LinkedList[int]:
    """A linked list of n perfect squares.

    >>> print(squares(10))
    (1 4 9 16 25 36 49 64 81 100)
    """
    def f(i):
        if i > n:
            return ()
        else:
            return Link(i*i, f(i+1))
    return f(1)
```

```
def low_squares(n) -> LinkedList[int]:
    """A linked list of n perfect squares
    that have a ones digit up to 5.

    >>> print(low_squares(10))
    (1 4 25 64 81 100 121 144 225 324)
    """
    def f(i, k):
        if k == n:
            return ()
        elif _________________:
            return Link(i*i, _____________ )
        else:
            return __________
    return f(1, 0)
```

What to square *How many found*

Building Linked Lists Recursively

```
type LinkedList[T] = Link[T] | tuple[()]

@dataclass
class Link[T]:
    first: T
    rest: LinkedList[T] = ()  # rest defaults to empty
```

Building a linked list using recursion often uses the recursive result as the rest

```
def some_function(...) -> LinkedList:
    ... Link(..., some_function(...)) ...
```

```
def squares(n) -> LinkedList[int]:
    """A linked list of n perfect squares.

    >>> print(squares(10))
    (1 4 9 16 25 36 49 64 81 100)
    """
    def f(i):
        if i > n:
            return ()
        else:
            return Link(i*i, f(i+1))
    return f(1)
```

```
def low_squares(n) -> LinkedList[int]:
    """A linked list of n perfect squares
    that have a ones digit up to 5.

    >>> print(low_squares(10))
    (1 4 25 64 81 100 121 144 225 324)
    """
    def f(i, k):
        if k == n:
            return ()
        elif i * i % 10 <= 5:
            return Link(i*i, f(i+1, k+1))
        else:
            return f(i+1, k)
    return f(1, 0)
```

What to square *How many found*

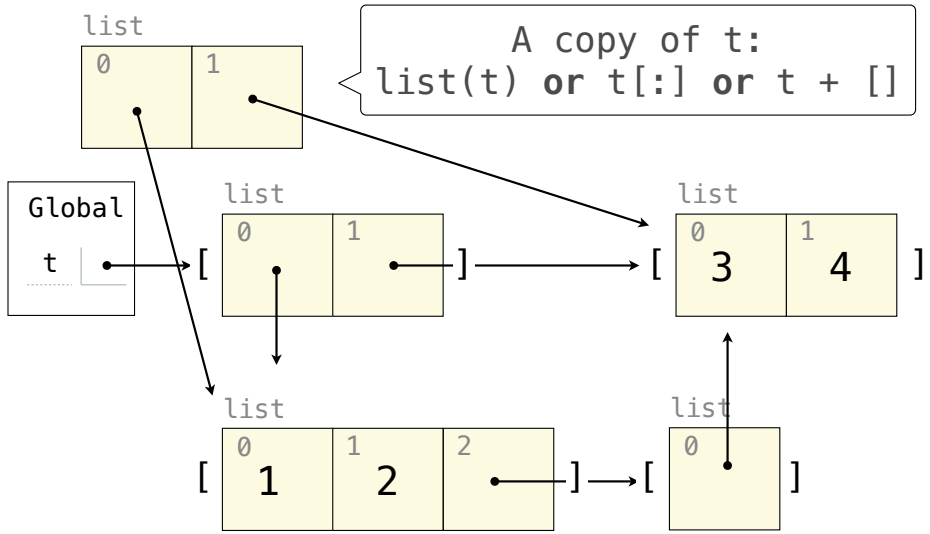
List Mutation

(Demo: Study Guide Examples)

https://pythontutor.com/cp/composingprograms.html#code=suits%20%3D%20%5B'coin',%20'string',%20'myriad',
%20'cup'%5D%0Asuits.pop%28%29%20%20%20%20%20%20%20%20%20%20%20%20%23%20remove%20%26%20return%20last%0Asuits.pop%281%29%20%20%20%20%20%20%20%20%20%20%20%20%23%20remove%20%26%20return%20index%201%0Asuits.remove%28'myriad'%29%20%20%20%23%20remove%20first%20match%0Asuits%0Asuits.append%28'spade'%29%20%20%20%20%23%20add%20one%20to%20the%20end%0Asuits.extend%28%5B'heart',
%20'club'%5D%29%20%20%23%20add%20all%20to%20end%0Asuits%5B0%5D%20%3D%20'diamond'%0Asuits.insert%280,%20'joker'%29%20%20%23%20add%20at%20index%0Asuits%0Aprint%28suits.append%28'x'%29%29%20%20%23%20append%20returns%20None%3B%20so%20do%20extend,
%20insert%20%26%20remove%0A%0As%20%3D%20%5B1,%202%5D%0As.append%28%5B3,%204%5D%29%0As.extend%28%5B5,%206%5D%29%0A%0At%20%3D%20s%0As%20%2B%3D%20%5B7%5D%20%20%20%20%23%20mutates%0As%20%3D%20s%20%2B%20%5B8%5D%20%20%23%20new%20list&cumulative=true&curInstr=17&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Lists in Lists in Lists in Environment Diagrams

```
t = [[1, 2], [3, 4]]
list(t)
t[0].append(t[1:2])
print(t)
```



The call to print(t) displays `[[1, 2, [[3, 4]]], [3, 4]]`

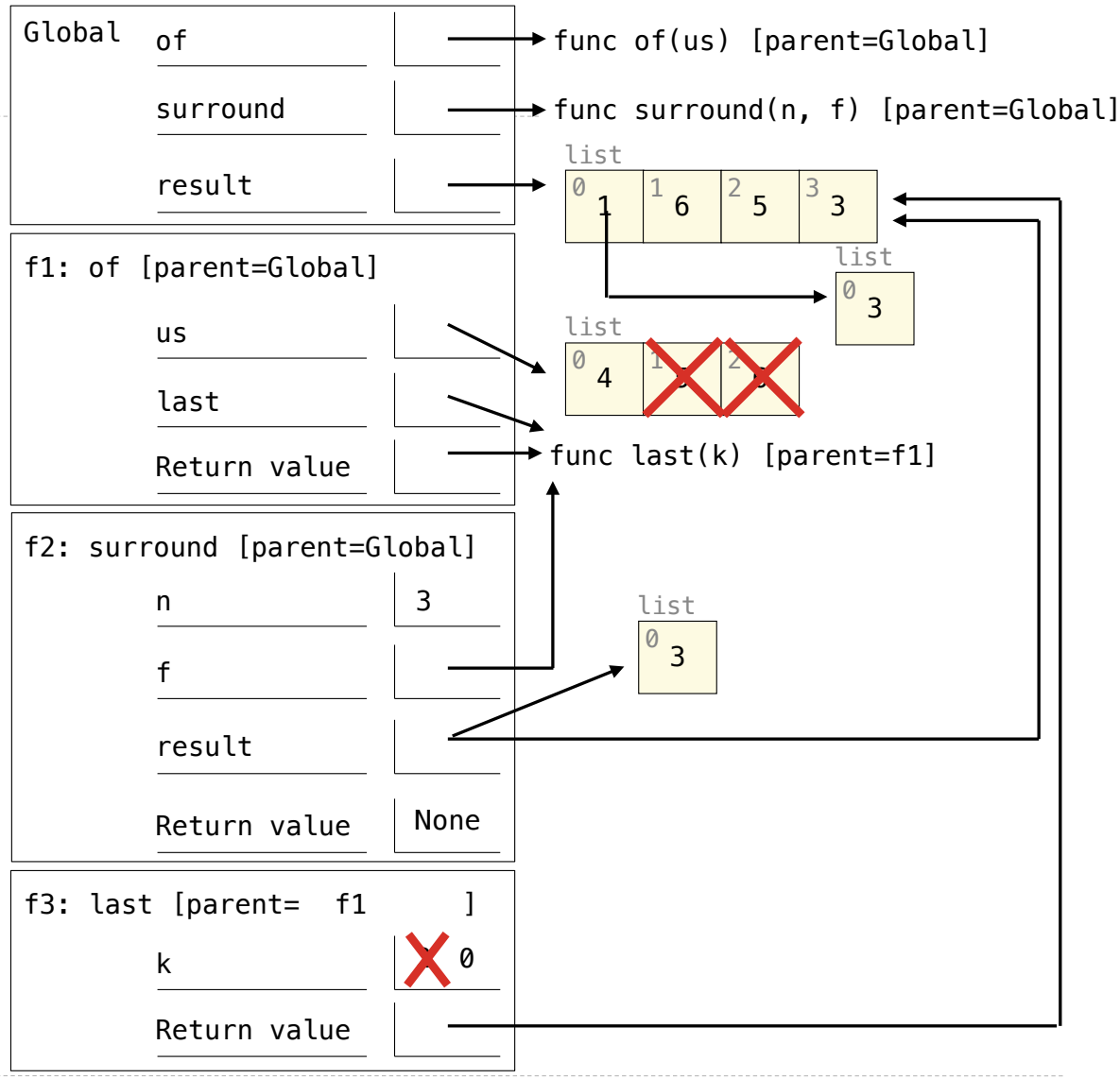
Fall 2022 Midterm 2 Question 2

```
def of(us):
    def last(k):
        "The last k items of us"
        while k > 0:
            result.append(us.pop())
            k = k - 1
        return result
    return last

def surround(n, f):
    "n is the first and last item of f(2)"
    result = [n]
    result = f(2)
    result[0] = [n]
    return result.append(n)

result = [1]
surround(3, of([4, 5, 6]))
print(result)
```

```
[[3], 6, 5, 3]
```



Spring 2023 Midterm 2 Question 1

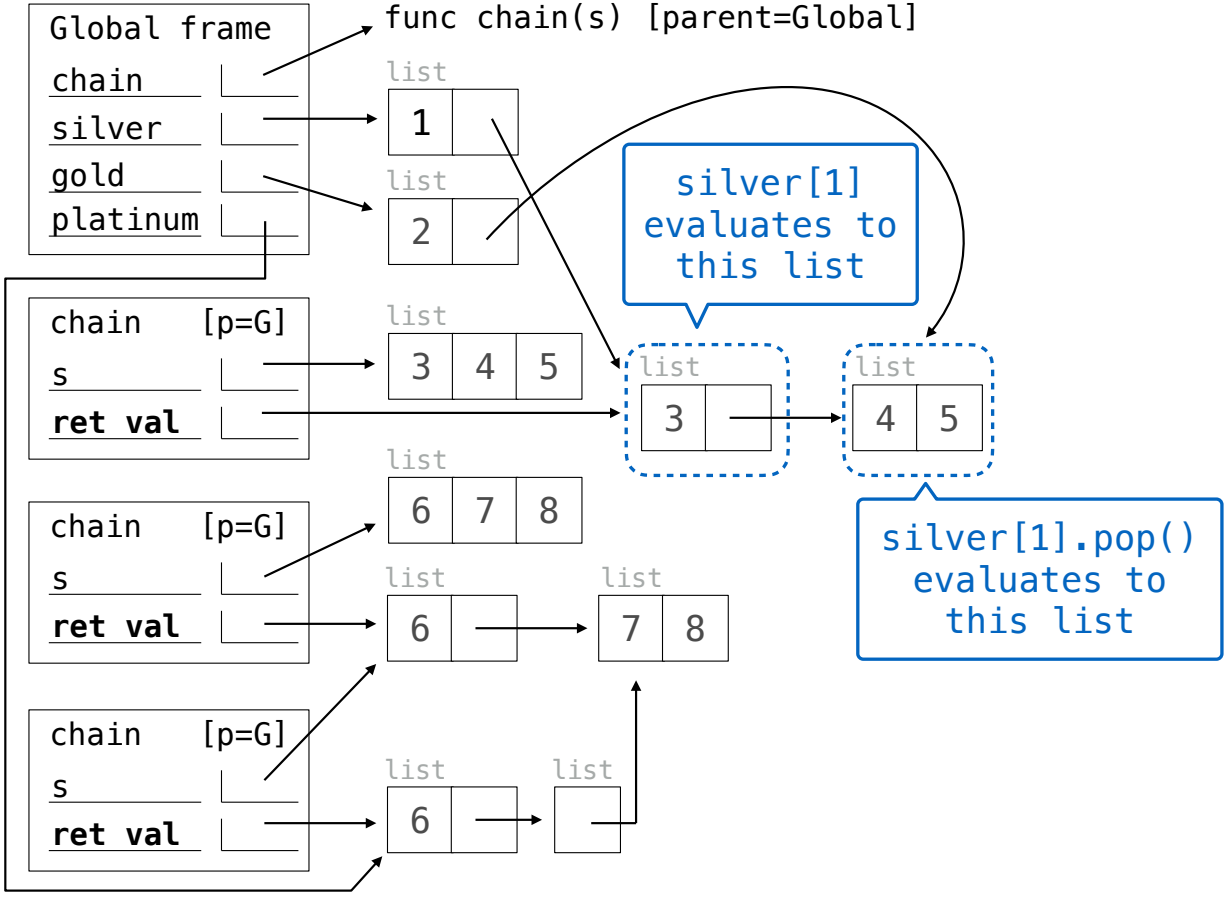
```
def chain(s):
    return [s[0], s[1:]]
silver = [2, chain([3, 4, 5])]
gold = [silver[0], silver[1].pop()]
silver[0] = 1
platinum = chain(chain([6, 7, 8]))
```

Reminder: s.pop() removes and returns the last item in list s.

```
>>> silver
[1, [3]]

>>> gold
[2, [4, 5]]

>>> platinum
[6, [[7, 8]]]
```



https://pythontutor.com/cp/composingprograms.html#code=def%20chain%28s%29%3A%0A%20%20%20%20return%20%5Bs%5B0%5D,%20s%5B1%3A%5D%5D%0Asilver%20%3D%20%5B2,%20chain%28%5B3,%204,%205%5D%29%5D%0Agold%20%3D%20%5Bsilver%5B0%5D,%20silver%5B1%5D.pop%28%29%5D%0Asilver%5B0%5D%20%3D%201%0Aplatinum%20%3D%20chain%28chain%28%5B6,%207,%208%5D%29%29%0A&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D