Sequences & Containers

Announcements

- Lab 3 due on Wednesday
- Hog Project due Thursday
  - Turn in by Wednesday for 1 EC point
  - Project Party on Wednesday and Thursday
- Grades for assignments have been released
  - Please carefully look through the Ed post
- Homework Recovery starts this week!
- July 4th – No lecture tomorrow
  - There will no makeup lecture, tutoring sections, OH, and discussion
- Midterm July 13th 7–9 pm
  - Exam alternations and accommodations, please fill out the form: go.cs61a.org/exam-alts

Sequences

Sequences

A sequence is an ordered collection of values

"hello world"

strings
sequence of characters

[1, “a”, 2, “b”]

lists
sequence of values of any data type

range(0, 10)

ranges
sequence of numbers

Strings

Strings are an Abstraction

Representing data:

```
'200'     '1.2e-5'     'False'     '[1, 2]'
```

Representing language:

```
"""And, as imagination bodies forth
The forms of things unknown, the poet's pen
Turns them to shapes, and gives to airy nothing
A local habitation and a name.
"""
```

Representing programs:

```
'curry = lambda f: lambda x: lambda y: f(x, y)'
```

String Literals Have Three Forms

```
>>> 'I am string!'
'I am string!'

>>> "I've got an apostrophe"
"I've got an apostrophe"

>>> '您好'
'您好'


>>> """The Zen of Python
claims, Readability counts.
Read more: import this."""
'The Zen of Python\nclaims, Readability counts.\nRead more: import this.'
```

Single-quoted and double-quoted strings are equivalent

A backslash "escapes" the following character

"Line feed" character represents a new line

Lists

Lists

A list is a **container** that holds a **sequence** of values of any data type

```
#empty list

>>> l = []
```

A list can hold **any** Python value, separated by commas

```
>>> names = [“Tim”, “Jordan”, “Noor”]

>>> funcs = [min, add, pow]

>>> years = [2023, 2019, 1999]

>>> apply = [pow, 2.0, 3, “eight”, “?”]
```

Creating Lists

```
>>> nums = [2, 81, 16]

>>> calc = [min(2, 3), square(9, 9), pow(2, 4)]



>>> nums

[2, 81, 16]

>>> calc

[2, 81, 16]

>>> list([2, 8, 16])

[2, 81, 16]
```

List Length

The **len** function computes the length of a list

```
#empty list

>>> l = []

>>> length = len(l)

>>> length

0


>>> names = [“Tim”, “Jordan”, “Noor”]

>>> len(names)

3
```

Indexing Lists

Each item in a list has an index, **starting with 0 -> len(list) - 1**

```
colors = [“Blue”, “Magenta”, “Yellow”, “Licorice”]

index       0         1          2          3
```

```
>>> len(colors)

4
```

You can access an items by putting square brackets [] around it

```
>>> colors[3]

“Licorice”

>>> getitem(colors, 2)

“Yellow”
```

[Demo]

Concatenation and Repetition

Lists can be added using the **+** operator

```
>>> colors = [“Blue”, “Magenta”, “Yellow”, “Licorice”]

>>> more = [“Orange”, “Lavender”]

>>> colors + more

[“Blue”, “Magenta”, “Yellow”, “Licorice”, “Orange”, “Lavender”]

>>> colors + more * 2

[“Blue”, “Magenta”, “Yellow”, “Licorice”, “Orange”, “Lavender”, “Orange”, “Lavender”]

>>> add(colors, mul(more, 2))

[“Blue”, “Magenta”, “Yellow”, “Licorice”, “Orange”, “Lavender”, “Orange”, “Lavender”]
```

List Slicing

Through slicing, a subpart of the list is obtained by passing in a <start index>, a non-inclusive <end index>, and [step size]

```
list[<start index>:<end index>:[step size]]
```

```
>>> s = [1, 3, 5, 7, 2, 4, 6, 8]
>>> s[0:3]
[1, 3, 5]
>>> s[0::2]
[1, 5, 2, 6]
>>> s[::-1]
[8, 6, 4, 2, 7, 5, 3, 1]
```

[Python Tutor Demo]

Nested Lists

Recall a list can contain any Python value, including another list!

```
>>> inventory = [[“Apples”, 2], [“Oranges”, 4], [“Onions”, 10]]

>>> len(inventory)

3
```

Be careful with len!

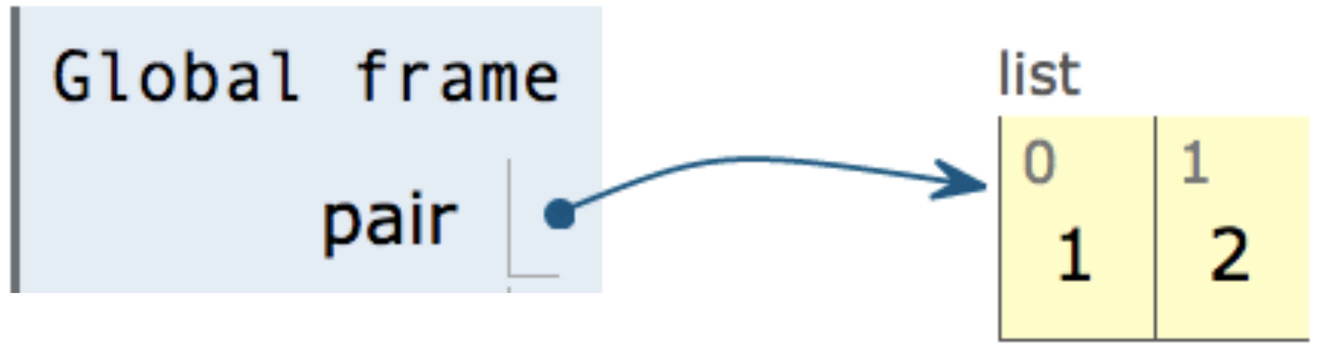
```
>>> inventory[1]

[“Oranges”, 4]

>>> inventory[1][1]

4
```

Box-and-Pointer Notation

Box-and-Pointer Notation in Environment Diagrams

Lists are represented as a row of index-labeled adjacent boxes, one per element

Each box either contains a primitive value or points to a compound value

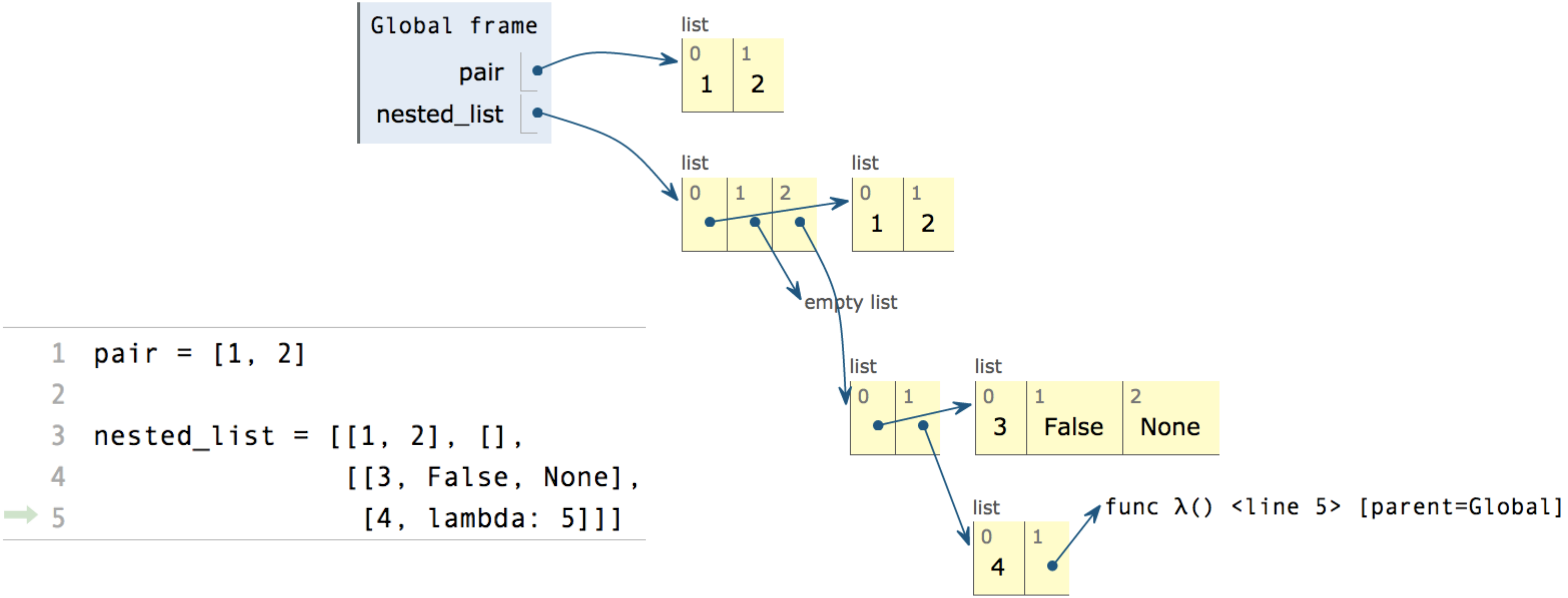


```
pair = [1, 2]
```

Box-and-Pointer Notation in Environment Diagrams

Lists are represented as a row of index-labeled adjacent boxes, one per element

Each box either contains a primitive value or points to a compound value



```
pair = [1, 2]

nested_list = [[1, 2], [],
               [[3, False, None],
                [4, lambda: 5]]]
```

pythontutor.com/composingprograms.html#code=pair%20%3D%20[1,%202]%0A%0Anested_list%20%3D%20[[1,%202],%20[],%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20[[3,%20False,%20None],
%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20[4,%20lambda%3A%205]]]&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=4

Strings as Sequences

Just as with lists, strings can also be indexed, concatenated, and sliced!

Recall, they are also sequences, therefore, all these ideas can be applied to them

(“Demo”)

Containers

Containers

Built-in operators for testing whether an element appears in a compound value: not in and in

```
>>> digits = [1, 8, 2, 8]

>>> 1 in digits
True

>>> 8 in digits
True

>>> 5 not in digits
True

>>> not(5 in digits)
True
```

(Demo)

Break

For Statements

For Statement Execution Procedure

```
for <name> in <expression>:
    <suite>
```

1. Evaluate the header <expression>, which must yield an iterable value (a sequence)

2. For each element in that sequence, in order:

   A. Bind <name> to that element in the current frame

   B. Execute the <suite>

(Demo)

Sequence Unpacking in For Statements

A sequence of fixed-length sequences

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

>>> same_count = 0
```

A name for each element in a fixed-length sequence

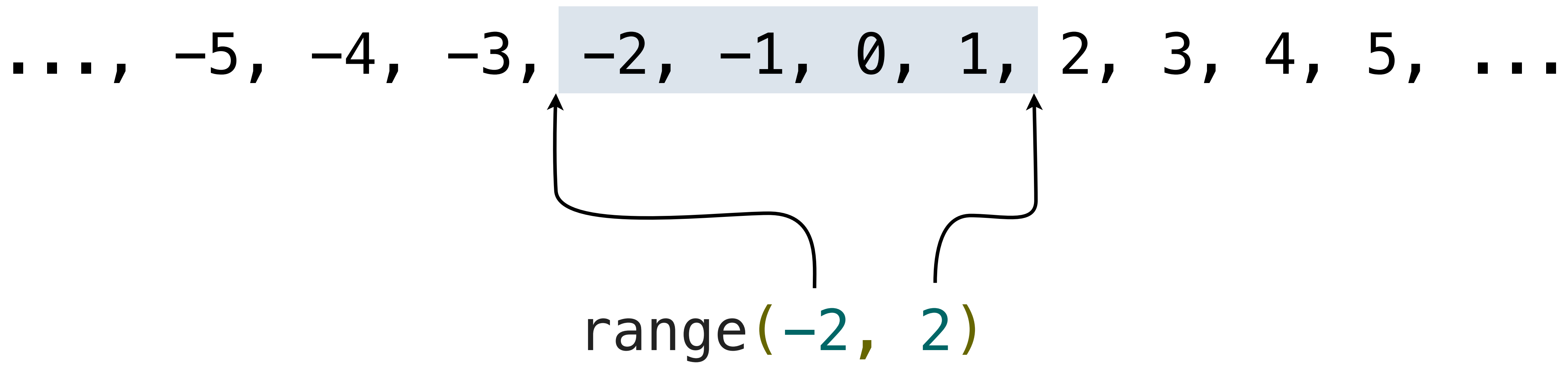
Each name is bound to a value, as in multiple assignment

```
>>> for x, y in pairs:
...     if x == y:
...         same_count = same_count + 1

>>> same_count
2
```

Ranges

The Range Type

The range function creates a sequence of consecutive integers.*



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

range(-2, 2)

range (<start>, <end>, [skip])

```
>>> list(range(-2, 2))
[-2, -1, 0, 1]
```

List constructor

```
>>> list(range(4))
[0, 1, 2, 3]
```

Range with a 0 starting value

* Ranges can actually represent more general integer sequences.

List Comprehensions

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'm', 'n', 'o', 'p']
>>> [letters[i] for i in [3, 4, 6, 8]]
```

```
['d', 'e', 'm', 'o']
```

List Comprehensions

```
[<map exp> for <name> in <iter exp> if <filter exp>]

Short version: [<map exp> for <name> in <iter exp>]
```

A combined expression that evaluates to a list using this evaluation procedure:

1. Add a new frame with the current frame as its parent

2. Create an empty *result list* that is the value of the expression

3. For each element in the iterable value of <iter exp>:

    A. Bind <name> to that element in the new frame from step 1

    B. If <filter exp> evaluates to a true value, then add the value of <map exp> to the result list

(Demo)

Aggregation

Several built-in functions take iterable arguments and aggregate them into a value

- **sum**(iterable[, start]) -> value

  Return the sum of an iterable (not of strings) plus the value
  of parameter 'start' (which defaults to 0). When the iterable is
  empty, return start.

- **max**(iterable[, key=func]) -> value
  max(a, b, c, ...[, key=func]) -> value

  With a single iterable argument, return its largest item.
  With two or more arguments, return the largest argument.

- **all**(iterable) -> bool

  Return True if bool(x) is True for all values x in the iterable.
  If the iterable is empty, return True.

Example: Promoted

First in Line

Implement **promoted**, which takes a sequence **s** and a one-argument function **f.** It returns a list with the same elements as **s,** but with all elements **e** for which **f(e)** is a true value ordered first. Among those placed first and those placed after, the order stays the same.

```
def promoted(s, f):
    """Return a list with the same elements as s, but with all
    elements e for which f(e) is a true value placed first.

    >>> promoted(range(10), odd)  # odds in front
    [1, 3, 5, 7, 9, 0, 2, 4, 6, 8]
    """
    return  [e for e in s if f(e)] + [e for e in s if not f(e)]
```

Dictionaries

```
{'Dem': 0}
```

Limitations on Dictionaries

Dictionaries are collections of key-value pairs

Dictionary keys do have two restrictions:

- A key of a dictionary **cannot be** a list or a dictionary (or any *mutable type*)
- Two **keys cannot be equal;** There can be at most one value for a given key

This first restriction is tied to Python's underlying implementation of dictionaries

The second restriction is part of the dictionary abstraction

If you want to associate multiple values with a key, store them all in a sequence value

Summary

- **Containers**, such as lists and dictionaries, can store **sequences** of values
- **List slicing** creates a new list
  - **list**[<start index>:<end index>:[step size]]
- We can iterate over sequences using **for statements**

```
for <name> in <expression>:
    <suite>
```

  - It is more concise than while statements, however, there are times when a while statement is more suitable
- **List comprehensions** allow us to return a new list using values of an existing list
  - In one line: **[**<map exp> for <name> in <iter exp> if <filter exp>**]**