Sequences

Announcements

Box-and-Pointer Notation

The Closure Property of Data Types

- A method for combining data values satisfies the *closure property* if:

  The result of combination can itself be combined using the same method

- Closure is powerful because it permits us to create hierarchical structures

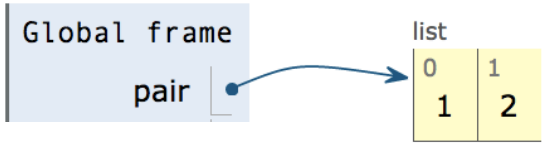
- Hierarchical structures are made up of parts, which themselves are made up of parts, and so on

Lists can contain lists as elements (in addition to anything else)

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

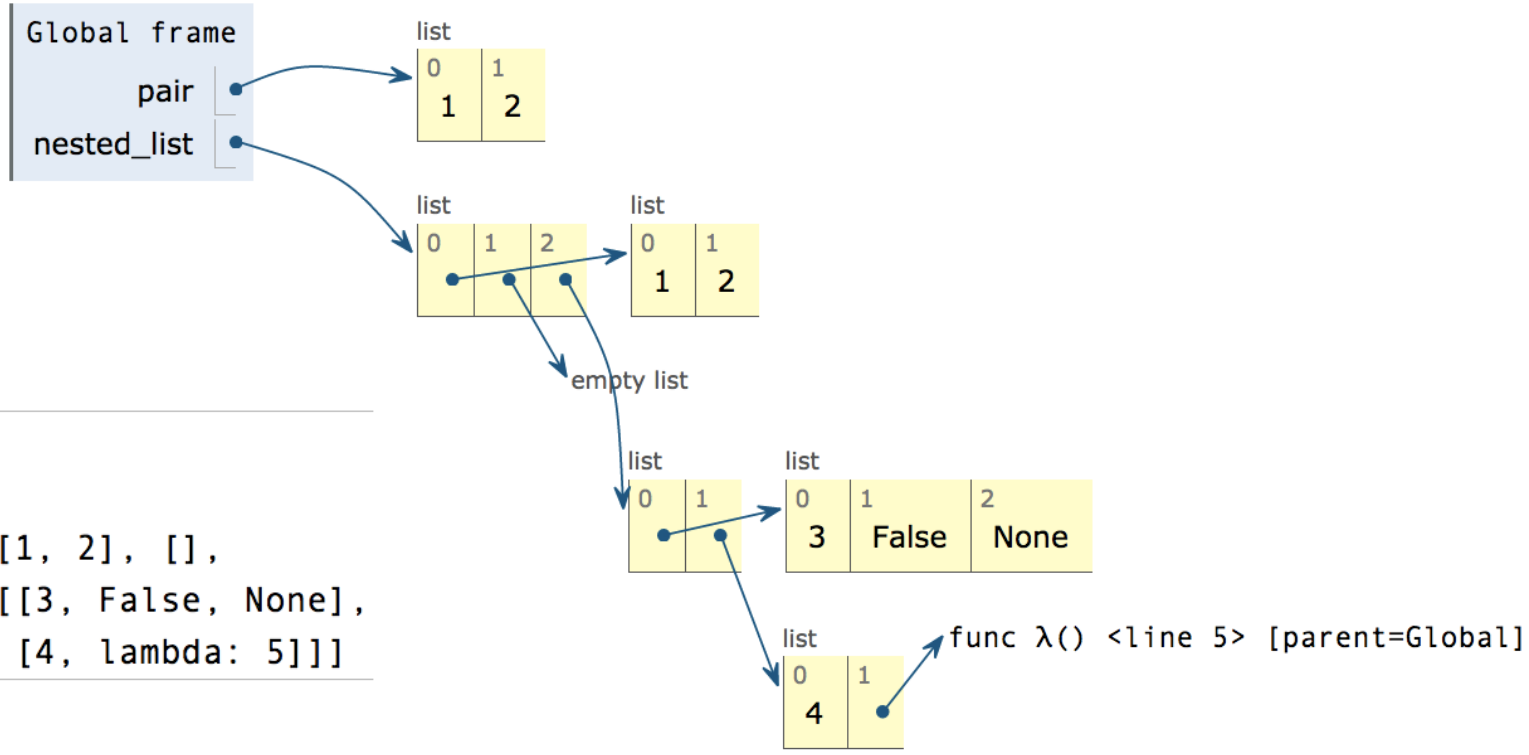



```
pair = [1, 2]
```

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




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

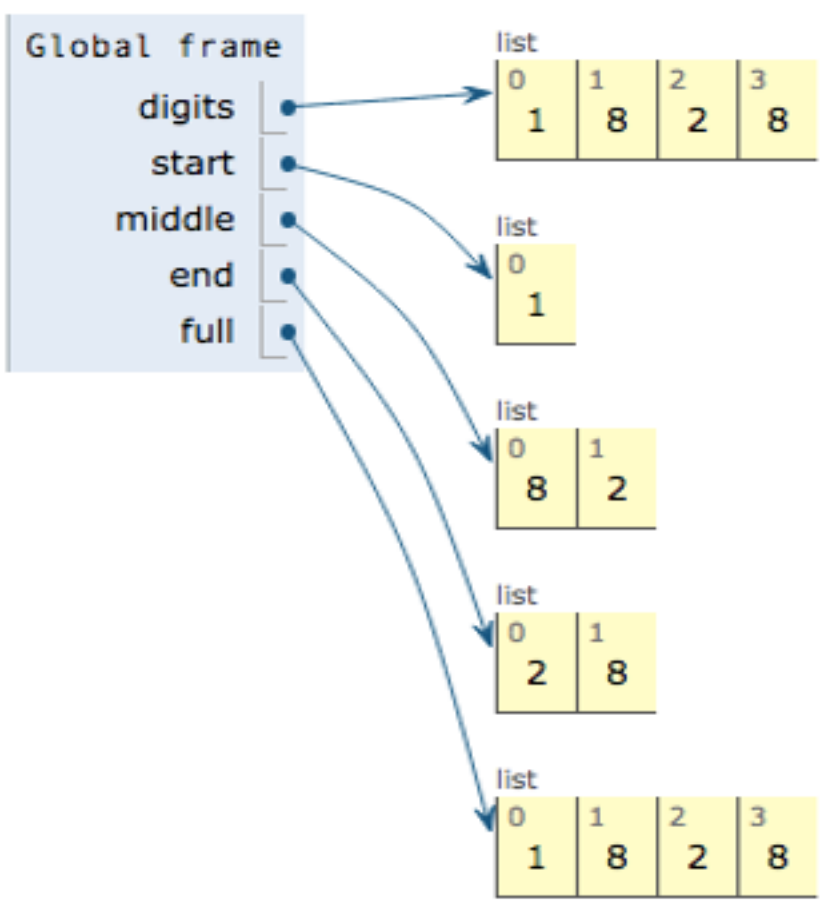

pythontutor.com/composingprograms.html#code=pair%20%3D%20[1,%202]%0A%0Anested_list%20%3D%20[[1,%202],%20[],%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20[[3,%20False,%20None],%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20[4,%20lambda%3A%205]]]&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curIn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

Slicing

(Demo)

Slicing Creates New Values

```
1  digits = [1, 8, 2, 8]
2  start = digits[:1]
3  middle = digits[1:3]
4  end = digits[2:]
5  full = digits[:]
```



pythontutor.com/composingprograms.html#code=digits%20%3D%20[1,%208,%202,%208]%0Astart%20%3D%20digits[%3A1]%0Amiddle%20%3D%20digits[1%3A3]%0Aend%20%3D%20digits[2%3A]%0Afull%20%3D%20digits[%3A]&cumulative%3Dtrue&curInstr%3D5&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=[]

Processing Container Values

Sequence 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.

Recursive Sums

Sum (recursively)

```
def mysum(L):
    if (L == []):
        return 0
    else:
        return L[0] + mysum( L[1:] )
```

```
mysum( [2, 4, 1, 5] )

2 + mysum( [4, 1, 5] )
    4 + mysum( [1, 5] )
        1 + mysum( [5] )
            5 + mysum( [] )
                0
```

```
# --- DRILL ---
# Write an iterative function that takes as input
# integer “n” and returns the sum of the first “n”
# integers: sum(5) returns 1+2+3+4+5
```

```
# --- DRILL ---
# Write an iterative function that takes as input
# integer “n” and returns the sum of the first “n”
# integers: sum(5) returns 1+2+3+4+5


def sum_iter(n):
    sum = 0
    for i in range(0,n+1):
        sum = sum + i

    return( sum )
```

```
# --- DRILL ---
# Write a recursive function that takes as input
# integer “n” and returns the sum of the first “n”
# integers: sum(5) returns 1+2+3+4+5
```

```
# --- DRILL ---
# Write a recursive function that takes as input
# integer “n” and returns the sum of the first “n”
# integers: sum(5) returns 1+2+3+4+5


def sum_rec(n):
    if( n == 0 ):
        return(0)
    else:
        return n + sum_rec(n-1)
```

Reversing a String

Reversing a List (recursively)

```
reverse("ward") = "draw"

reverse("ward") = reverse("ard") + "w"

    reverse("ard") = reverse("rd") + "a"

        reverse("rd") = reverse("d") + "r"

            reverse("d") = "d"
```

Reversing a List (recursively)

```
reverse("ward") = "draw"

reverse("ward") = reverse("ard") + "w"

    reverse("ard") = "d" + "r" + "a"
```

Reversing a List (recursively)

```
def reverse(s):
    if len(s) == 1:
        return s
    else:
        return reverse(s[1:]) + s[0]
```