

Sharing, mutability, and immutability

Michael Ernst

CSE 140

University of Washington

Copying and mutation

```
list1 = ["e1", "e2", "e3", "e4"]
list2 = list1
list3 = list(list1)    # make a copy; also "list1[:]"
print list1, list2, list3
list1.append("e5")
list2.append("e6")
list3.append("e7")
print list1, list2, list3
list1 = list3
list1.append("e8")
print list1, list2, list3
```

Variable reassignment vs. object mutation

- Reassigning a variable does not change (mutate) any object
 - Always done via the syntax
`myvar = expr`
- Mutating (changing) an object does not change any variable binding
 - Two syntaxes:
`left_expr = right_expr`
`expr.method(args...)`
 - Examples:
`mylist[3] = myvalue`
`mylist.append(myvalue)`

New and old values

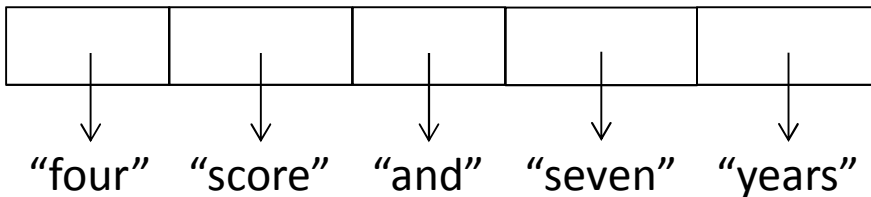
- Every expression evaluates to a value
 - It might be a new value
 - It might be a value that already exists
- A **constructor** evaluates to a **new** value
 - [3, 1, 4, 1, 5, 9]
 - [3, 1, 4] + [1, 5, 9]
 - [3, 1, 4, 1, 5, 9]
- An **access** expression evaluates to an **existing** value
 - mylist = [[3, 1], [4, 1]]
 - mylist[1]
- What does a function call evaluate to?

An aside: List notation

- Possibly misleading notation:

| | | | | |
|--------|---------|-------|---------|---------|
| "four" | "score" | "and" | "seven" | "years" |
|--------|---------|-------|---------|---------|

- More accurate, but more verbose, notation:



Object identity

- An object's identity never changes
- Its value (the thing it represents) may change

```
mylist = [1, 2, 3]
otherlist = mylist
mylist.append(4)
mylist is otherlist           ⇒ True
mylist == [1, 2, 3, 4] ⇒ True
mylist is [1, 2, 3, 4] ⇒ False
```


The object identity test “**is**” is rarely used

Object type and variable type

- An object's type never changes
- A variable can get rebound to a value of a different type
- A type indicates:
 - what operations are allowed
 - the set of representable values

Aside: how did tuples get their name?

- singleton
- pair
- double
- triple
- quadruple
- quintuple
- sextuple
- septuple
- octuple
- nonuple
- decuple

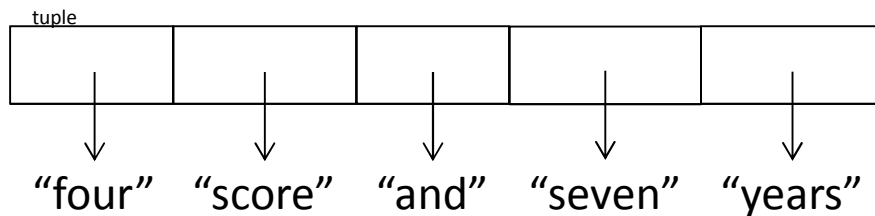
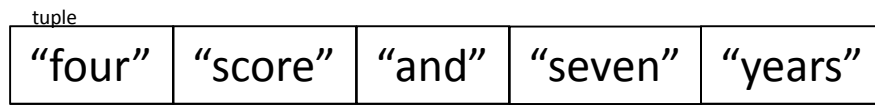


Notice that the last 5 letters in these words are always **tuple**

New datatype: tuple

A tuple represents an ordered sequence of values

Example:



Tuple operations

Constructors

- Literals: Just like lists, but round the square brackets

`("four", "score", "and", "seven", "years")`

- Also `(3, 1) + (4, 1) => (3, 1, 4, 1)`, etc.

Queries

- Just like lists

Mutators

- **None!**

Immutable datatype

- An immutable datatype is one that doesn't have any functions in the third category:
 - Constructors
 - Queries
 - Mutators: **None!**
- Immutable datatypes:
 - int, float, boolean, string, function, tuple, *frozenset*
- Mutable datatypes:
 - list, dictionary, set

Not every value may be placed in a set

- Set elements must be immutable values
 - int, float, bool, string, *tuple*
 - *not*: list, set, dictionary
- Goal: only set operations change the set
 - after “`myset.add(x)`”, `x in myset` \Rightarrow True
 - `y in myset` always evaluates to the same valueBoth conditions should hold until `myset` is changed
- Mutable elements can violate these goals

```
list1 = ["a", "b"]
```

```
list2 = list1
```

```
list3 = ["a", "b"]
```

```
myset = { list1 }
```

\Leftarrow Hypothetical; actually illegal in Python

```
list1 in myset  $\Rightarrow$  True
```

```
list3 in myset  $\Rightarrow$  True
```

```
list2.append("c")
```

```
list1 in myset  $\Rightarrow$  ???
```

```
list3 in myset  $\Rightarrow$  ???
```

Not every value is allowed to be a key

- Keys must be immutable values
 - int, float, bool, string, *tuple*
 - *not*: list, set, dictionary
- Goal: only dictionary operations change the keyset
 - after “`mydict[x] = y`”, `mydict[x] ⇒ y`
 - if `a == b`, then `mydict[a] == mydict[b]`These conditions should hold until `mydict` is changed
- Mutable keys can violate these goals

```
list1 = ["a", "b"]
list2 = list1
list3 = ["a", "b"]
mydict = {}
mydict[list1] = "z"
mydict[list3] ⇒ "z"
list2.append("c")
mydict[list1] ⇒ ???
mydict[list3] ⇒ ???
```

← Hypothetical; actually illegal in Python

Python's *Data Model*

- Everything is an *object*
- Each object has:
 - an *identity*
 - Never changes
 - Test with `is` (but you rarely need to do so)
 - a *type*
 - Never changes
 - a *value*
 - Can change for *mutable* objects
 - Cannot change for *immutable* objects
 - Test with `==`

Identity

```
>>> A = [1]
>>> B = [1]
>>> A == B
True
>>> A is B
False
>>> C = A
>>> A is C
????
```

```
>>> A = [1]
>>> B = [1]
>>> A == B
True
>>> A is B
False
```

```
>>> conjugations = {
"see":["saw", "sees"],
"walk":["walked", "walks"]
"do":["did", "does"]
"be":["was", "is"]
}
>>> conjugations["see"]
???
```

| |
|-------------------------|
| conjugations["walk"][1] |
|-------------------------|

```
???
```

| |
|----------------------------|
| conjugations["walk"][1][0] |
|----------------------------|

```
???
```

```
>>> [word[0] for word in conjugations["be"]]
???
```

| |
|--|
| [pair for pair in conjugations.items()][0] |
|--|

```
???
```

| |
|---|
| [(pair[0][0], pair[1][0][0]) for pair in conjugations.items()][1] |
|---|

```
???
```

| |
|--|
| {pair[0]:pair[1] for pair in conjugations.items()} |
|--|

```
???
```


Mutable and Immutable Types

```
>>> def increment(uniqewords, word):  
...     """increment the count for word"""  
...     uniqewords[word] = uniqewords.setdefault(word, 1) + 1
```

```
>>> mywords = dict()  
>>> increment(mywords, "school")  
>>> print mywords  
{'school': 2}
```

```
>>> def increment(value):  
...     """increment the value???"  
...     value = value + 1  
>>> myval = 5  
>>> increment(myval)  
>>> print myval  
5
```

Tuples are immutable

Lists are mutable

```
def updaterecord(record, position, value):  
    """change the value at the given position"""  
    record[position] = value
```

```
mylist = [1,2,3]  
mytuple = (1,2,3)  
updaterecord(mylist, 1, 10)  
print mylist  
updaterecord(mytuple, 1, 10)  
print mytuple
```

Mutable and Immutable Types

- Immutable
 - numbers, strings, tuples
- Mutable
 - lists and dictionaries

Note: a set is mutable, but a *frozenset* is immutable

Mutable and Immutable Types

```
>>> def increment(uniqewords, word):  
...     """increment the count for word"""  
...     uniqewords[word] = uniqewords.setdefault(word, 1) + 1
```

```
>>> mywords = dict()  
>>> increment(mywords, "school")  
>>> print mywords  
{'school': 2}
```

```
>>> def increment(value):  
...     """increment the value???"  
...     value = value + 1  
>>> myval = 5  
>>> increment(myval)  
>>> print myval  
5
```