Intro to Python (Class 3)

Ben Bettisworth

Section 1

Collections

Collections

Aside from the "primitive" types in python, there are types which are made up of other types. We will call these types "collections"

Built-in Collections

- Lists
- Tuples
- Sets
- Dictionaries

Collections as Values

In addition to primitive values, collections can also contain other collections.

Example

```
l1 = [1,2,3]
l2 = [1, 2, l1]
```

Section 2

Lists

Lists

Lists are an ordered series of *values*. The values can be any type.

Examples

```
list_a = [1,2,3,4,5]
list_b = ["foo", "bar"]
list_c = [1, 2, 3, "foo", "bar"]
list_d = [None, None, None]
```

Creating Lists

There are a couple of ways to create a list.

- The first is with the syntax `[]`
- The second is with the keyword `list`

```
a = []
b = list()
```

Accessing Items in a List

Items in a list can be accessed using `a[x]`.

Example

```
list_a = [1,2,3,4,5]
print(list_a[2])
```

Output

```
3
```

Accessing Items in a List

Python is a *zero-indexed* language. This means the first item in a list has *index 0*.

Example

```
list_a = ["chocolate", "vanilla", "strawberry"]
print("first item is (index 0)", list_a[0])
print("last item is (index 2)", list_a[2])
```

Output

```
first item is (index 0) chocolate
last item is (index 2) strawberry
```

Adding Items to a List

To add items to a list, there are 2 operations:

- `append`
- `insert`

Appending Items to a List

Items can be appended (added to the end) of a list with the function `list.append(x)`

Code

```
a = []
a.append(1)
a.append(2)
print(a)
```

Output

```
[1, 2]
```

Inserting Items to a List

Items can be inserted into a list with the function `list.insert(i, x)`

Code

```
a = []
a.insert(0, 1)
a.insert(1, 2)
print(a)
```

Output

```
[1, 2]
```

Removing Items from a List

There are 2 ways of removing items from a `list`.

- `list.remove(x)`
- `list.pop(x)`

Removing Items from a List

`list.remove(x)` removes the first *value* equal to `x`.

Example

```
a = [1, 2, 1]
a.remove(1)
print(a)
```

Output

```
[2, 1]
```

Removing Items from a List

`list.pop(x)` removes the item with index `x`, and returns its value. If `x` is omitted, it removes and returns the last value.

Example

```
a = [1, 2, 1]
a.pop(1)
print(a)
```

Output

```
[1, 1]
```

Tuples

Tuples are immutable lists. They are made with either the `tuple(x,y)` function, or with the notation `t = (a,b)`.

Example

```
t = (1,2,3)
t[1]
```

Output

```
2
```

Tuples are Immutable

Immutable means that the tuple cannot be changed. This means that operations like `append` and `pop` do not exist for tuples.

Example

```
t = (1,2,3)
t.append(4)
```

Output

```
AttributeError: 'tuple' object has no attribute
  'append'
```

Exercise

Write a function that takes a list (or tuple) of tree items and adds them together.

Exercise

(A possible) Solution

```
def sum3(l):
  return l[0] + l[1] + l[3]

print(sum3([1,2,3]))
```

Output

```
6
```

Exercise

Write a function that takes the first item of the list, and adds it to the back.

Exercise

(A Possible) Solution

```
def add_first(l):
  tmp = l.copy()
  l.append(l[0])
  return l

l = ["chocolate", "strawberry"]
print(add_first(l))
```

Output

```
['chocolate', 'strawberry', 'chocolate']
```

Exercise

(Another Possible) Solution

```
def add_first(l):
  tmp = l[0]
  l.append(tmp)

l = ["chocolate", "strawberry"]
add_first(l)
print(l)
```

Output

```
['chocolate', 'strawberry', 'chocolate']
```

Section 3

References

References

There are two *kinds* of values in python: immutable, and references. Primitives (`int`, `float`, `str`) are immutable, while lists are references.

Rules to remember

- Variables assigned to the same immutable value will *not* change other variables when modified
- Variables assigned to the same reference value *will* change other when modified.

Example

Example

```
a = "choco"
b = a
b += "basil"
print(a)
```

Output

```
'choco'
```

Example

```
a = ["choco", "vanilla"]
b = a
b.append("basil")
print(a)
```

Output

```
['choco', 'vanilla',
  'basil']
```

Argument Passing

Function arguments in python are *pass by assignment*. Like the previous slide, this means that *some* values can be modified by the function.

Example

```
def foo(a):
  a = a * 2
b = 2
foo(b)
print(b)
```

Output

```
2
```

Example

```
def foo(a):
  a.append(100)
b = [1,2,3]
foo(b)
print(b)
```

Output

```
[1, 2, 3, 100]
```

Methods

You might have noticed that `a.append(1)` has a new kind of syntax.

- `append` is a *method*
- Methods are like functions, but they operate on values.
- E.g. `append` modifies the list `a`.
- Some methods don't modify the value
    - `count(x)`

Methods

You should think of methods as acting on the value which they are called on.

Example

```
a.append(1)
```

Here, `append` is called on `a`, and adds the value `1` to the end of `a`.

Section 4

Conditional Evaluation

Conditional Evaluation

- Conditional evaluation is used by programs to do something *some of the time*.
- An example is to check if a number is negative before a calculation.
  - $(-1)^{\frac{1}{2}} = i$
- Another is to check the input from a user.

if/else Statements

Conditional expressions are python normally use the `if` keyword. If the condition is true, they execute the *body* of the `if`. Otherwise, they execute the body of the `else`.

Example

```
if a < 0:
  print("The square root of a negative \
        number is imaginary!")
else:
  print("The square root is", a**(0.5))
```

Conditional Statements

Conditional statements are produced with operators. They can be read as asking the question "is this statement true?" Examples of conditional operators are.

- `a == b` (is equal)
- `a != b` (is *not* equal)
- `a < b`/`a > b` (is less than / greater than)
- `a <= b`/`a >= b` (is less than / greater than or equal)
- `a is b` (identity)
- `a in b` (inclusion)

Exercise

Write a function that takes some number x, and informs the user if it is divisible by some number k. Remember that a number n is divisible by k if and only if the remainder of division is 0. That is

```
n % k == 0
```

Extension: If the number is divisible by k, print the other factors.

Exercise

(A Possible) Solution

```python
def divides(x, k):
  if x % k == 0:
    print(f"{x} is divisble by {k}")
  else:
    print(f"{x} is not divisble by {k}")

divides(10, 2)
divides(7, 3)
```

Output

```
10 is divisble by 2
7 is not divisble by 3
```

Exercise

Write a function which informs the user that a list contains the number 2.

Extension: tell the user how many times it contains the number 2.

Exercise

Code

```
def has2(l):
  if 2 in l:
    print("Your list has 2!")
  else:
    print("Your list doen't have 2 :(")

has2([2,4,5,1,2,5])
```

Output

```
Your list has 2!
```

Composite conditional expresions

Conditional expressions can be combined with `and` / `or`.

Example

```
if a == 0 or a == 3:
  print("a is zero or three!")
```

Example

```
if a > 0 and a < 5:
  print("a is between 0 and 5")
```

Precedence

The operators and / or have a precedence order, and comes before or.

Code

```
a = 12
if a == 0 and a % 2 == 0 or a % 3 == 0 and a == 6:
  print("first statement is true")
if a == 0 or a % 2 == 0 and a % 3 == 0 or a == 6:
  print("second statement is true")
```

Chaining if / else Statements

If many conditional statements are required in a series, then we can use `elif`.

Example

```
if a == 0:
  print("A is nothing!")
elif a == 1:
  print("A is singular!")
else:
  print("I don't know what A is :(")
```

Here, the second conditional is only checked if the first one is false.