galvanize-logo

Python 2

Diving into data structures, control flow, and logical operators

Learning Objectives

1. **Containers** - Discuss and work with all major data structures: sets, lists, tuples, dictionaries

1. **Control Flow** - Effectively use 'for' loops, 'while' loops, and comprehensions

1. **Logical Operators** - Effectively carry out logical operations

1. **Debugging** - Perform basic debugging operations

Containers

Collection data types in Python

- A `list` in Python is a ordered and changeable.
- A `tuple` in Python is ordered and unchangeable
- A `set` in Python ordered and does not use and index
- A `frozenset` in Python has the samep properties of a set except its valued are changeable
- A `dict` uses key-value pairs and is unordered, changeable and indexed.

- lists, tuples and dictionaries allow for duplicates but sets do not

list comprehensions and a review of built-in types

- We have already covered built-in types (https://docs.python.org/3/library/stdtypes.html)
- They are immutable

immutable objects are objects whose value can't change once created

In [33]:
```
# illustrate the covered data types by introducing
data_types = [True, 4, 4.0, 1 + 2j, 'abc', None]

# determine the different data types
print([type(data_type) for data_type in data_types])
print(list(map(type, data_types)))

# proper way to check for the data type
print(isinstance(data_types[0],bool))
```

```
[<class 'bool'>, <class 'int'>, <class 'float'>, <class 'complex'>, <class 'st
r'>, <class 'NoneType'>]
[<class 'bool'>, <class 'int'>, <class 'float'>, <class 'complex'>, <class 'st
r'>, <class 'NoneType'>]
True
```

Mutable vs immutable

- Some collection classes are mutable: `set`, `list` and `dict`
- Others are immutable: `tuple`, `string`, `frozenset`

In [12]:
```
an_iterable = ["A","G","C","T","U"]

# to check if an object is an iterable you can turn it into an iterator
as_iterable = iter(an_iterable)
print(next(as_iterable))
print(next(as_iterable))

# Show how to create most containers
a_tuple = ("A","G","C","T","U"),        # tuple(an_iterable)
a_list = ["A","G","C","T","U"]          # list(an_iterable)
a_set = {"A","G","C","T","U"}           # set(an_iterable)
a_frozen_set = frozenset(a_list)
```

```
A
G
```

Tuples

These collections which are ordered and unchangeable. They are denoted with round brackets.

In [74]:
```
# instantiating and indexing
course = ("Accelerated Python", "2020", "Fall", 65)
print("Name = ", course[0])
print("Last element = ", course[-1])

# unpacking
name, year, semester, students = course
print("Year =", year)

# use the * to capture items during tuple unpacking
name, *semester_id, students = course
print("Semester_id = ", semester_id)
print("Semester_id = ", "-".join(semester_id))

# the * has a number of other uses as well
print(*course,sep="---")
```

```
Name =  Accelerated Python
Last element =  65
Year = 2020
Semester_id =  ['2020', 'Fall']
Semester_id =  2020-Fall
Accelerated Python---2020---Fall---65
```

Sets

These collections which contain **unique** members and do not use an index

```
In [83]:
draw1 = {"🂡", "🂢","🂫"}
draw2 = {"🃚", "🃛","🂫"}
print("draw1", draw1)
print("draw2", draw2)

# adding to the set
draw1.add("🃗")
print("modified draw1", draw1)

# set operations
print("cards in both draws: ", draw1.intersection(draw2))
print("cards only in draw1", draw1.difference(draw2))
```

```
draw1 {'🂫', '🂡', '🂢'}
draw2 {'🂫', '🃛', '🃚'}
modified draw1 {'🂫', '🂡', '🂢', '🃗'}
cards in both draws:  {'🂫'}
cards only in draw1 {'🂡', '🂢', '🃗'}
```

The humble dictionary

- As of Python 3.6 dictionaries maintain order
- a set of key:value pairs
- dictionaries are indexed by keys, which can be any immutable type
- Average complexity of dictionary lookups to O(1) through the use of a hash function (https://en.wikipedia.org/wiki/Hash_function)

In [90]:

```
# different ways to instantiate a dict
a_dict = {"a":1, "b":2, "c":3}
a_dict = dict(a=1, b=2, c=3)
a_dict = dict([("a", 1), ("b", 2), ("c", 3)])
a_dict = {}
a_dict["a"] = 1
a_dict["b"] = 2
a_dict["c"] = 3

print(a_dict)
print("keys", list(a_dict.keys()))
print("values", list(a_dict.values()))
print("items", list(a_dict.items()))
```

```
{'a': 1, 'b': 2, 'c': 3}
keys ['a', 'b', 'c']
values [1, 2, 3]
items [('a', 1), ('b', 2), ('c', 3)]
```

A more realistic example

Index data borrowed from *Pattern Recognition and Machine Learning* by CM Bishop

```
In [85]: prml_index = {"marginal likelihood": [162, 165],
                       "Markov Chain Monte Carlo": [537],
                       "Markov rondom field": [84, 360, 383]}

         print(a_dict["a"])
         print(prml_index["Markov Chain Monte Carlo"])
```

```
1
[537]
```

Is it indexable?

- Not all data containers have an index
- dictionaries require
- Sets are efficient in part because they do not have the overhead needing to index
- An introduction to the humble `for` loop

In [46]:

```python
for data_container in [a_tuple, a_list, a_set, a_frozen_set, a_dict]:
    dtype = type(data_container).__name__

    try:
        foo = data_container[-1]
        print("Objects of type '{}' are indexable".format(dtype))
    except:
        print("Objects of type '{}' are NOT indexable".format(dtype))
```

```
Objects of type 'tuple' are indexable
Objects of type 'list' are indexable
Objects of type 'set' are NOT indexable
Objects of type 'frozenset' are NOT indexable
Objects of type 'dict' are NOT indexable
```

Identity, equality, and memory

You will get very comfortable with the different operators (https://en.wikipedia.org/wiki/Operator_(computer_programming))

- the == operator checks for **equality**
- the `is` operator checks **identify**

The operator module (https://docs.python.org/3/library/operator.html) enables a more functional programming style option.

Containers - Check for understanding

QUESTION 1

Given the containers you can you think of a convenient way to look up the nearest city given Showing results for longitude and latitude? Assume that they data has been pre-calculated.

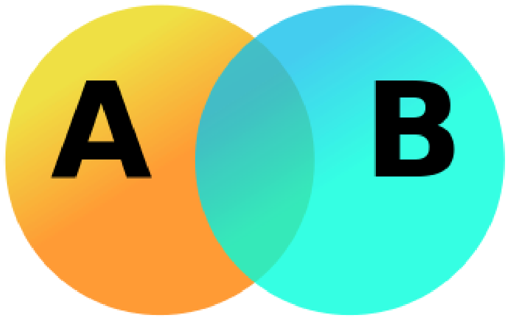
Latitude	Longitude	City
48.9	2.4	Paris, France
-12.1	-77.0	Lima, Peru

```
In [ ]:
```

Containers - Check for understanding

QUESTION 2

I have constructed word clouds from two different surveys. I want to create a Venn diagram, which container would be the most appropriate?



source: wiki

Control Flow

if elif-else

In [95]:
```
x, y = 9,10

# simple input validation
if not isinstance(x,float):
    x = float(x)
elif not instance(y,float):
    y = float(y)

# use logical operators to control the flow
if (x > y):
    print(x, '>', y)
elif (x == y):
    print(x, 'equals', y)
else:
    print('Either', x, '<', y, 'or x and y are not orderable')
```

```
Either 9.0 < 10 or x and y are not orderable
```

while loops

In [101]:
```
## simple while loop
i = 0
while (i < 4):
    print(i)
    i += 1
```

```
0
1
2
3
```

In [100]:
```
# while True as a condition ensures that the code runs until it's broken
i = 0
while True:
    print(i)
    if i > 4:
        break
    i += 1
```

```
0
1
2
3
4
5
```

for loops

Remember these data?

```
In [108]: jan_sales = [1834., 1918.,  812., 1680., 2492., 2776., 2390., 2297.]
          feb_sales = [2148., 1745., 2190., 1863., 2589., 2345., 2724., 2239., 2785., 148
          3., 2038., 2021.]
          mar_sales = [1968., 1718., 1634., 2126., 1252., 2538., 2837., 1223., 2034., 161
          1., 2791.]
          apr_sales = [2496., 2733.,  706., 2386., 3382., 1844., 1440., 2594., 1978., 202
          3., 2559., 1577.]
          may_sales = [2832., 1681., 1954., 1801., 2294., 1732., 1638., 1949., 2676., 232
          9., 2370.]
          jun_sales = [2335., 2538., 2186., 2186., 2622., 2564., 1269., 3124., 1286., 168
          9., 2627., 1345.]
          jul_sales = [1651., 1957.,  853., 2229., 2990., 3148., 2917.,  952., 1583., 244
          7., 2491.]
          aug_sales = [2520., 2540., 1756., 1562., 972., 2258., 1413., 1779., 2503., 2860.
          ]
          sep_sales = [1827., 2003., 1349., 1858., 1370., 1076., 2897., 2238.,   91., 195
          1., 2509., 2933.]
          oct_sales = [1273., 3169., 1192., 2219., 2195., 3157., 2912., 2012.,  722.,  92
          2.]
          nov_sales = [1827., 2003., 1349., 1858., 1370., 1076., 2897., 2238.,   91., 195
          1., 2509., 2933.]
          dec_sales = [2200., 2460., 1260., 3157., 2912., 2012.,  722.,  922.]

          sales_data = [jan_sales, feb_sales, mar_sales, apr_sales, may_sales, jun_sales,
          jul_sales, aug_sales]
          sales_data += [sep_sales, oct_sales, nov_sales, dec_sales]
```

In [118]:
```
cogs_percentage = 0.6

# for loop
monthly_gross_revenues, monthly_net_revenues = [],[]
for month in sales_data:
    monthly_gross_revenues.append(sum(month))
    monthly_net_revenues.append(sum(month) - (sum(month) * cogs_percentage))
```

In [119]:
```
# list comp
monthly_gross_revenues = [sum(m) for m in sales_data]
monthly_net_revenues = [sum(m) - (sum(m) * cogs_percentage) for m in sales_data]


print([round(m) for m in monthly_gross_revenues])
print([round(m) for m in monthly_net_revenues])
```

```
[16199, 26170, 21732, 25718, 23256, 25771, 23218, 20163, 22102, 19773, 22102,
15645]
[6480, 10468, 8693, 10287, 9302, 10308, 9287, 8065, 8841, 7909, 8841, 6258]
```

Comprehensions

There is a lot more we can do with comprehensions.

In [128]:
```
import string

# reverse a dict
d1 = {i:l for i,l in enumerate(string.ascii_lowercase)}
d2 = {v: k for k, v in d1.items()}
print(list(d1.items())[:5])
print(list(d2.items())[:5])
```

```
[(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd'), (4, 'e')]
[('a', 0), ('b', 1), ('c', 2), ('d', 3), ('e', 4)]
```

In [105]:
```
# What built-in functions are available in python again?
", ".join([x for x in dir(__builtin__)  if x.islower() and not x.startswith('_
_')])
```

Out[105]:
```
'abs, all, any, ascii, bin, bool, breakpoint, bytearray, bytes, callable, chr,
classmethod, compile, complex, copyright, credits, delattr, dict, dir, displa
y, divmod, enumerate, eval, exec, filter, float, format, frozenset, get_ipytho
n, getattr, globals, hasattr, hash, help, hex, id, input, int, isinstance, iss
ubclass, iter, len, license, list, locals, map, max, memoryview, min, next, ob
ject, oct, open, ord, pow, print, property, range, repr, reversed, round, set,
setattr, slice, sorted, staticmethod, str, sum, super, tuple, type, vars, zip'
```

Recap

Topic	Learning Objective
Containers	Discuss and work with all major data structures: sets, lists, tuples, dictionaries
Control Flow	Effectively use 'for' loops, 'while' loops, and comprehensions
Logical Operators	Effectively carry out logical operations

Resources

- Official Python Tutorial (https://docs.python.org/3/tutorial/)

```
In [ ]:
```