Data Types Cheat Sheet

Numbers: integers and floats

- Integers don't have a decimal place.
- Floats have a decimal place.
- Math mostly works the way it does on a calculator, and you can use parentheses to override the order of operations.

Math: addition, subtraction, multiplication

addition: 2 + 2
subtraction: 0 - 2
multiplication: 2 * 3

Math: division

```
>>> 4 / 2
2
>>> 1 / 2
0
```

- Integer division produces an integer. You need a number that knows about the decimal point to get a decimal out of division:

```
>>> 1.0 / 2
0.5
>>> float(1) / 2
0.5
```

Types

```
>>> type(1)
<type 'int'>
>>> type(1.0)
<type 'float'>
```

Strings

- Strings are bits of text, and contain characters like numbers, letters, whitespace, and punctuation.
- String are surrounded by quotes.
- Use triple-quotes (""") to create whitespace-preserving multi-line strings.

```
```

```
>>> "Hello"
'Hello'
```

String concatenation

```
>>> "Hello" + "World"
HelloWorld
>>> "Hello" + "World" + 1
Traceback (most recent call last):
  File "", line 1, in
TypeError: cannot concatenate 'str' and 'int' objects
>>> "Hello" + "World" + str(1)
'HelloWorld1'
```

Printing strings

```
>>> print "Hello" + "World"
HelloWorld
```

```
>>> name = "Jessica"
>>> print "Hello " + name
Hello Jessica
```

```
>>> print """In 2009,
...     The monetary component of the Nobel Prize
...         was US .4 million."""
In 2009,
    The monetary component of the Nobel Prize
        was US .4 million.
```

Types

```
>>> type("Hello")
<type 'str'>
```

Booleans

- There are two booleans, `True` and `False`.
- Use booleans to make decisions.

Containment with 'in' and 'not in'

```
```

```
>>> "H" in "Hello"
True
>>> "a" not in ["a", "b", "c"]
False
```

Equality

- `==` tests for equality
- `!=` tests for inequality
- `<`, `<=`, `>`, and `>=` have the same meaning as in math class.

```
>>> 0 == 0
True
>>> 0 == 1
False
```

```
"a" != "a"
```

```
"a" != "A"
```

Use with if/else blocks

- When Python encounters the `if` keyword, it evaluates the expression following the keyword and before the colon. If that expression is `True`, Python executes the code in the indented code block under the if line. If that expression is `False`, Python skips over the code block.

```
temperature = 32
if temperature > 60 and temperature < 75:
    print "It's nice and cozy in here!"
else:
    print "Too extreme for me."
```

Types

```
>>> type(True)
<type 'bool'>
>>> type(False)
<type 'bool'>
```

Lists

- Use lists to store data where order matters.

- Lists are indexed starting with 0.

List initialization

```
>>> my_list = []
>>> my_list
[]
>>> your_list = ["a", "b", "c", 1, 2, 3]
>>> your_list
['a', 'b', 'c', 1, 2, 3]
```

Access and adding elements to a list

```
>>> len(my_list)
0
>>> my_list[0]
Traceback (most recent call last):
  File "", line 1, in
IndexError: list index out of range
>>> my_list.append("Alice")
>>> my_list
['Alice']
>>> len(my_list)
1
>>> my_list[0]
'Alice'
>>> my_list.insert(0, "Amy")
>>> my_list
['Amy', 'Alice']
```

```
>>> my_list = ['Amy', 'Alice']
>>> 'Amy' in my_list
True
>>> 'Bob' in my_list
False
```

Changing elements in a list

```
>>> your_list = []
>>> your_list.append("apples")
>>> your_list[0]
'apples'
>>> your_list[0] = "bananas"
>>> your_list
['bananas']
```

```

```

Slicing lists

```
>>> her_list = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
>>> her_list[0]
'a'
>>> her_list[0:3]
['a', 'b', 'c']
>>> her_list[:3]
['a', 'b', 'c']
>>> her_list[-1]
'h'
>>> her_list[5:]
['f', 'g', 'h']
>>> her_list[:]
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```

Strings are a lot like lists

```
>>> my_string = "Hello World"
>>> my_string[0]
'H'
>>> my_string[:5]
'Hello'
>>> my_string[6:]
'World'
>>> my_string = my_string[:6] + "Jessica"
>>> my_string
'Hello Jessica'
```

- One big way in which strings are different from lists is that lists are mutable (you can change them), and strings are immutable (you can't change them). To "change" a string you have to make a copy:

```
>>> h = "Hello"
>>> h[0] = "J"
Traceback (most recent call last):
  File "", line 1, in
TypeError: 'str' object does not support item assignment
>>> h = "J" + h[1:]
>>> h
'Jello'
```

Types

```

```

```
>>> type(my_list)
<type 'list'>
```

Dictionaries

- Use dictionaries to store key/value pairs.
- Dictionaries do not guarantee ordering.
- A given key can only have one value, but multiple keys can have the same value.

Initialization

```
>>> my_dict = {}
>>> my_dict
{}
>>> your_dict = {"Alice" : "chocolate", "Bob" : "strawberry", "Cara" :
"mint chip"}
>>> your_dict
{'Bob': 'strawberry', 'Cara': 'mint chip', 'Alice': 'chocolate'}
```

Adding elements to a dictionary

```
>>> your_dict["Dora"] = "vanilla"
>>> your_dict
{'Bob': 'strawberry', 'Cara': 'mint chip', 'Dora': 'vanilla', 'Alice':
'chocolate'}
```

Accessing elements of a dictionary

```
>>> your_dict["Alice"]
'chocolate'
>>> your_dict.get("Alice")
'chocolate'
```

```
>>> your_dict["Eve"]
Traceback (most recent call last):
  File "", line 1, in
KeyError: 'Eve'
>>> "Eve" in her_dict
False
>>> "Alice" in her_dict
True
>>> your_dict.get("Eve")
>>> person = your_dict.get("Eve")
>>> print person
```

```
None
>>> print type(person)
<type 'NoneType'>
>>> your_dict.get("Alice")
'coconut'
```

Changing elements of a dictionary

```
>>> your_dict["Alice"] = "coconut"
>>> your_dict
{'Bob': 'strawberry', 'Cara': 'mint chip', 'Dora': 'vanilla', 'Alice':
'coconut'}
```

Types

```
>>> type(my_dict)
<type 'dict'>
```
