How to python and variable types

How to run python

Option 1: Use the command line interpreter

```
$ python
>>> print(“hello world”)
“hello world”
>>> # this is a comment, python
ignores these
...
>>> exit() # how you leave the
interpreter
```

How to run python

Option 2: execute a script, always have the extension of your script be ".py"

```
$ cat test.py
print(“hello world”)
$ python test.py
hello world
```

Variable types

Numbers: 2, 3.14, 94504050

Strings: “hello”, ‘100’, “““ this
string has multiple lines”””

Lists: [1 , 2], [“this”, “is”, “a list”]

Dictionary: {0 : 1, 1 : 2}, {“key” : “value” }

Numbers

There are two types of numbers: int or integers, cannot store fractional values, and float or floating point numbers can store any number in a huge range.

```
# Number examples
num_1 = 1 # int
num_2 = 1.5 # float
num_3 = 1. # float
```

Number Operations

```
>>> 2 + 3 # int addition
5

>>> 2. + 3. # float addition
5.0

>>> 2. + 3 # still float addition since
one of the two number is a float
5.0

>>> 5 / 2 # int division, there can be
no fractional values in int, thus is
rounded down
2

>>> 5. / 2. # float division
2.5
```

```
>>>1 == 1 # with ints it is always
possible to check to see if they are
same value with "==" operator
True

>>>1 > 2
False

>>>2 >= 2 # greater or equal
True

>>>a = 5
>>>a += 5 # same as a = a + 5
>>>print(a)
10
```

All operations: +, -, *, /, //, %, >, <, ==, !=, >=, <=, *=, +=, -=, /=

Take 5 mins to try some operations, are there any surprises?

Strings

Strings store text or any series of characters

```
# can use 'single quotes' or "double 
quotes" or """triple quotes"""
# triple quotes and span multiple 
lines
str_1 = "Hello World"
str_2 = '100'
str_3 = """sdgwerg@#$@
       #$%343 """
```

String Operations

```
>>>"hello" + "world" # concatenation or
string addition
'helloworld'

>>>"hello"*3 # repetition or string
multiplication
'hellohellohello'

>>>"hello"[0] # indexing, getting a
specific character
'h'

>>>"hello"[-1] # indexing from the end
‘o’

>>>len("hello") # get the length or
size
5
```

```
>>>"hello" < "jello" # comparison,
compares each letter at a time, the
letter that later in the alphabet is
'larger'
True

>>>2 >= 2 # greater or equal
True

>>>a = 5
>>>a += 5 # same as a = a + 5
>>>print(a)
10

>>>”hello”.upper() # all characters
become uppercase
’HELLO’
```

```
All operations: +, *, >, <, ==, !=, >=, <=, *=, +=
.lower(), .upper(), .count(), .index()
Take 5 mins to try some operations, are there any surprises?
```

Lists

A series of elements, initialized with []

```
list_1 = [] # empty list
list_2 = [1, 2] # a list of two
elements, first is 1 and second is 2
list_3 = [“element_1”, 1, [“sub”,
”list]] # lists can be composed on
any of the other types even other
lists!
```

List Operations

```
>>>[] # empty list
[]

>>>len([]) # get the number of elements
in a list
0

>>>a = [99, "hello", ["nested",
"list"]] # lists can store any value,
even other lists!
>>>print(a) # prints the values of the
list that is stored in variable 'a'
[99, 'hello', ['nested', 'list']]

>>>a = [0, 1]
>>>b = [2, 3]
>>>a + b # list addition, joins them
together
[0, 1, 2, 3]
```

```
>>>a * 3 # list multiplication creates
with duplicate elements
[0, 1, 0, 1, 0, 1]

>>>a = range(5) # creates a list with
elements 0 to 4
>>>print(a)
[0, 1, 2, 3, 4]

>>>a.append(5) # append adds a element
to end of a list
>>>print(a)
[0, 1, 2, 3, 4, 5]

>>>a.pop() # removes last element for
list and returns it
5
>>>print(a)
[0, 1, 2, 3, 4]
```

All operations: +, *, ==, [], .append(), .clear(), .copy(), .count(), .extend(), .index(), .insert(), .pop(), .remove(), .reverse()

Take 5 mins to try some operations, are there any surprises?

Dictionaries

A series of 'paired' elements, one is a unique key and one is a value. Instead of using indices like in lists to get values you can specify any value for a key. Dictionaries are defined using {} brackets

```
d = {'key1' : 'value1’}
d_1 = {0 : 1, 1: 2}
```

Dictionary Operations

```
>>>d = {'key1' : 'value1', 'key2' :
'value2'}
>>>print(d)
{'key2': 'value2', 'key1': 'value1'}

>>>len(d) # get number of items
2

>>>d['key1'] # get specific value by a
key
'value’

>>>d['key3'] # error for an invalid key
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'key3'
```

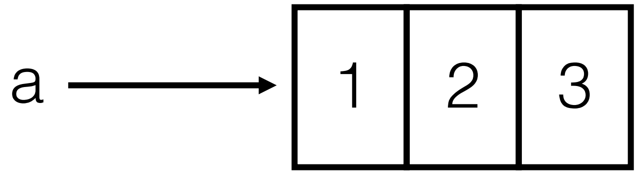
```
>>>d['key3'] = 'value3' # add new
key/value to dictionary
>>>d['key3']
'value3'

>>>d['key3'] = 'new_value' # override
previous value

>>>del d['key'] # removes key/value
pair from dictionary

>>>d.keys() # returns a list of keys in
the dictionary
['key2', 'key1']

>>>d.values() # returns a list of
values in the dictionary
['value2', 'value1']
```

All operations: `.append()`, `.clear()`, `.copy()`, `.pop()`, `.get()`, `.keys()`, `.values()`, `.items()`

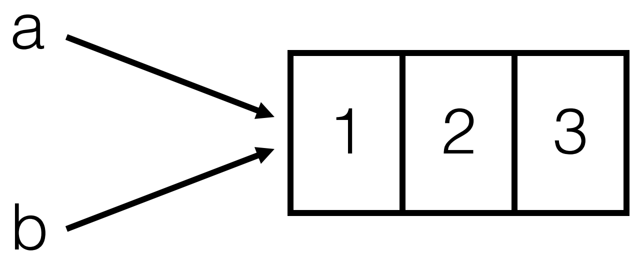
Take 5 mins to try some operations, are there any surprises?

Reference Semantics

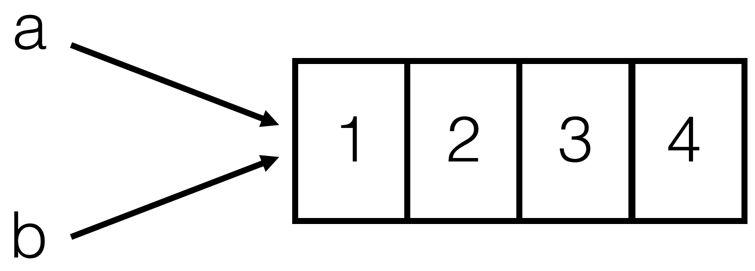
```
a = [1, 2, 3]
```



```
b = a
```



```
a.append(4)
```



Reference Semantics

```
>>>a = [1, 2, 3, 4]
>>>b = a  # now b is another reference or
'name' for the same list
>>>a.append(4)
>>>print(b)
>>>[1, 2, 3, 4]

# to create a new list instead of giving the
same list a new name
# any of these will work
>>> b = a.copy()
>>> b = list(a)
>>> b = a[::]
```

Introduction to control structures and indentation

if statements allow decisions to be made based on the resultant values of conditionals

```
if 5 > 2: #5 is greater than 2 so will
execute statement within the if
statement. Statements within the if
statement are tabbed
...     print("this is always true")
...
this is always true

>>>if 5 > 2:
...print("wrong")
  File "<stdin>", line 2
    print("wrong")
        ^
IndentationError: expected an indented
block

>>>if 5 > 2:
...  print("correct tabbing")
correct tabbing
```

```
# if statements can be paired with else
statements,
>>>if 2 > 5:
...   print("will not execute")
...else:
...   print("will execute")
...
will execute

# if statements can also include elif
statements which are a combination of a
if and else statement.
>>>num = 5
>>>if num == 4:
...  print("num is 4")
...elif num == 5:
...  print("num is 5")
...else:
...  print("num is something else")
num is 5
```

Introduction to getting input from users

Most programs you have used require you to input some data, the simplest way to do this in python is with the `input` function

```
>>>name = input("enter your name: ") # I
will enter Joe when prompted
enter your name: Joe
>>>print(name)
Joe
```

Practice Problem

Write a program that takes a number as an input and prints out whether it is odd or even