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?

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