In [114]:
```
print('Hello World!!')
```

```
Hello World!!
```

This is text.

UCO Carpentry Workshop

Python - Day 1

Nathalia Graf-Grachet

Goals: ¶

1. Jupyter notebook
2. Variables & data types
3. Storing multiple values - data structures
4. For loops and conditionals
5. Functions
6. Automation
7. Errors and help

Jupyter

Setup http://swcarpentry.github.io/python-novice-gapminder/setup/ (http://swcarpentry.github.io/python-novice-gapminder/setup/)

Jupyter http://swcarpentry.github.io/python-novice-gapminder/01-run-quit/index.html (http://swcarpentry.github.io/python-novice-gapminder/01-run-quit/index.html)

https://www.oreilly.com/ideas/what-is-jupyter (https://www.oreilly.com/ideas/what-is-jupyter)

Jupyter is one way to write code with Python. It's really nice because we can save all the commands and all notes as a notebook. If you work in a lab, you probably have a lab notebook that you keep with all your protocols, your modifications to the protocol, results, discussion, etc. Jupyter notebook is the same but for your code. For those of you wanting to get into programming and you need to keep notes of your scripts, jupyter is a great way to keep scripts, code, and notes.

Things to point out in jupyter:

Header bar == notebook name, last time saved (checkpoint)
Tool bar == icons

- name/rename the notebook
- save
- adding cells below
- cell layout = where code goes, where output shows
- different cells (code and markdown) example of python code
  example of markdown (headings, bold, italic, bullet points, lists, line breaks (2 spaces, or < br >))
- when writing code, run cell
- click or double-click on a cell (blue color - 'selected'), then click inside the code box (turns green - can write)
- go back, re-write code and run the same cell

8/12/2019

Level 2 Heading

Level 3 Heading

This text is *italicized*.
This text is **bold**.
This text is ***bold and italicized***.

- item 1
- item 2

-nospaceitem3

1. ordered item1
2. ordered item2
   A. ordered subitem
   B. ordered subitem2
3. and another ordered item.
4. number does not matter

In Jupyter, you can freely mix in mathematical expressions using the MathJax subset of Tex and LaTeX.

I like some math equations:

$$\left(\sum_{k=1}^{n} a_k b_k\right)^2 \leq \left(\sum_{k=1}^{n} a_k^2\right)\left(\sum_{k=1}^{n} b_k^2\right)$$

and this one:

$$P(E) = \binom{n}{k} p^k (1-p)^{n-k}$$

You can even mix math and formatted text in a paragraph. Just write expressions like so: `$\expression$` and they render like this: $\sqrt{3x-1}+(1+x)^2$

In [115]:
```
# This is a comment, Python does not read as code...
# It's useful to write little comments in your code... helps you remem
ber what you wrote (:
```

Let's define some Python terms...

Python is a **object** oriented language... (huh... okay)

Object: pretty much all the *stuff*. It's any data, any group of data, any arrangement of data, files, functions written by us of by python (built-in functions).

```
Data will be stored in variables
```

Data types: different types of data = string, integer, float, list, dictionary, etc.

```
'string'
25
3.14
```

Functions: a ***little programs*** that do things with the data, such as print, get maximum value, etc. Could be built-in, or created by user.

```
function_name(your_data_object)
```

Methods: are functions, but methods are functions that belong to an object. Different objects will have specific methods/functions. We'll work with strings and lists today -- string methods won't work with lists, even if the method returns the same information.

```
your_data_object.method_name(maybe_an_option)
```

Variables

First of all: what's a variable?

A variable is an imaginary box that we store values or data in. If you have a file with data, which could be a table with entries and numbers or genomic data, python will only process that data if it's in variable.

Use = to assign a value to a variable.

Variable names on the left -- values on the right.

Variable name is arbitrary -- no spaces, letters (lowercase, case-sensitive), digits (cannot start with a digit), but choose a name that is meaningful to you

In [118]:
```
age = 31
first_name = 'Nathalia'
```

You must create the variables before you can work with it.

Variables created in this notebook will persist between cells.

Use the built-in function `print(x)` to print the values of your variables.
`x` is your variable.

In [119]:
```
print(age) # Did you notice that Jupyter automatically writes the clos
ing paranthesis?!
```

```
31
```

In [120]:
```
print(first_name) # remember the tab auto complete?! It works here too
!
```

```
Nathalia
```

In [121]:
```
age # interactive
```

Out[121]:
```
31
```

In [122]:
```
first_name
```

Out[122]:
```
'Nathalia'
```

Data Types

FYI, I'll be creating a lof of variables. Some will have similar names, which can get confusing. I apologize in advance! If something is not clear, or if it got confused, make sure you stop me!!

1. String

In [479]:
```
fruit='mango'
```

In [20]:
```
fruits='mango and orange'
```

In [22]:
```
print(fruit)
print(fruits)
```

```
mango
mango and orange
```

2. Integers == whole numbers

In [245]:
```
age=31
```

In [24]:
```
tires=4
```

3. Floats == decimals, fractions

In [36]:
```
fraction_a = 3/8
```

In [26]:
```
a_decimal=0.35
```

In [25]:
```
a_exponent=1e-10
```

4. Boolean

In [354]:
```
True
```

Out[354]: True

In [355]:
```
False
```

Out[355]: False

In [65]:
```
True == True
```

Out[65]: True

In [66]:
```
True == False
```

Out[66]: False

In [67]:
```
False == False
```

Out[67]: True

Object types, data structures

1. List

A list stores many values in a single variable.

Create a list by writing values separated by `,` within `[]`

In [475]:
```
groceries = ['bread','cheese','tomato']
```

In [476]:
```
groceries
```

Out[476]: ['bread', 'cheese', 'tomato']

2. Dictionary

A dictionary stores values as key value pairs.

Create a dictionary by writing `'key':'value'` pairs separated by `,` within `{ }`

In [111]:
```
jewlery = {'necklace' : 3.45, 'bracelet': 7e4 }
```

3. Set

A set is a collection of **unique** elements/values.

Create a set by writing `set(list)`

In [106]:
```
rainfall_list=[1,1,1,3,1.5,5]
rainfall_set=set(rainfall_list)
rainfall_set
```

Out[106]: {1, 1.5, 3, 5}

Find out the object/data type

`type(x)` is built-in function that returns the type of data stored in a variable.

In [57]:
```
type(fruit)
```

Out[57]: str

In [58]:
```
type(tires)
```

Out[58]: int

In [59]:
```
type(exponent)
```

Out[59]: float

In [60]:
```
type(groceries)
```

Out[60]: list

In [107]:
```
type(jewlery)
```

Out[107]: dict

In [62]:
```
type(rainfall_set)
```

Out[62]: set

In [358]:
```
type(True)
```

Out[358]: bool

Built-in functions:

`print(x)`

`print()` can take strings and variables.
This function will automatically put a single space between arguments, and a new line at the end.

In [70]:
```
print('My age is',age)
```

```
My age is 31
```

`len(x)`

In [480]:
```
g=len(fruit) # number of characters in the string
print(g)
```

```
5
```

As nested function:

In [481]:
```
print(len(groceries)) # how many items in a list
```

```
3
```

Calculations

In [73]:
```
a=1.5
b=4.5
```

In [74]:
```
c=a+b
print(c)
```

```
6.0
```

In [75]:
```
d=c/a
print(d)
```

```
4.0
```

In [276]:
```
e=c/b
print(e)
```

```
1.3333333333333333
```

In [300]:
```
9//2 # floor division, returns quotient
```

Out[300]: 4

In [301]:
```
9%2 # modulus, returns remainder
```

Out[301]: 1

In [303]:
```
print("{0:.2f}".format(e))
round(e,2)
```

```
1.33
```

Out[303]: 1.33

Import `math` module for more mathematical functions

In [283]:
```
import math
```

If you need to use the constant `pi`, you can:

In [312]:
```
math.pi
```

Out[312]: 3.141592653589793

You must call the package name first. `pi` is a variable that had the constant value stored in it.

The value is only available when you call `math.pi`

If you call `pi` Python thinks it's a variable you created.

In [319]:
```
math.tau
```

Out[319]: 6.283185307179586

In [320]:
```
math.pi*2
```

Out[320]: 6.283185307179586

In [321]:
```
round(math.pi*2,2)
```

Out[321]: 6.28

Operations

Can't add numbers and strings... even if the string is a number.

`a=1` is different than `a='1'`

Must convert numbers to strings, or vice-versa

In [267]:
```
g='3'
```

In [271]:
```
print(1 + g) # run cell, than do conver int(g)
```

```
---------------------------------------------------------------------
-------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-271-02c36435927a> in <module>
----> 1 print(1 + g)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

In [269]:
```
print(str(g) + '2')
```

```
32
```

In [305]:
```
print('-' * 100)
```

```
----------------------------------------------------------------------
--------------------------------
```

Re-assigning value to a variable

```
age =31
```

In [253]:
```
age
```

Out[253]: 31

In [249]:
```
age=age+5
```

In [250]:
```
age
```

Out[250]: 36

31 was replaced by 31+5

In [254]:
```
age=31
```

In [255]:
```
my_age=age-6
my_age
```

Out[255]: 25

In [256]:
```
age
```

Out[256]: 31

STRING

Index `[x]`

In [93]:
```
first_name
```

Out[93]: 'Nathalia'

In [94]:
```
first_name[0]
```

Out[94]: 'N'

In [95]:
```
first_name[3]
```

Out[95]: 'h'

STRING

Slicing `[start_index : end_index]`

`end_index` not inclusive

In [99]:
```
first_name[0:3]
```

Out[99]: 'Nat'

In [100]:
```
first_name[0:8]
```

Out[100]: 'Nathalia'

In [101]:
```
nickname=first_name[0:3]+'y'
print(nickname)
```

```
Naty
```

LIST

Index and slicing

In [150]:
```
fruits[2]
```

Out[150]: 'lime'

In [151]:
```
fruits[0:3]
```

Out[151]: ['banana', 'orange', 'lime']

In [152]:
```
fruit_basket=fruits[0:3]
print(fruit_basket)
```

```
['banana', 'orange', 'lime']
```

In [153]:
```
fruits
```

Out[153]:
```
['banana', 'orange', 'lime', 'mango']
```

In [154]:
```
fruit_basket
```

Out[154]:
```
['banana', 'orange', 'lime']
```

LIST

Appending items to a list with `.append(x)` method

In [155]:
```
fruits.append('tomato')
```

In [156]:
```
fruits
```

Out[156]:
```
['banana', 'orange', 'lime', 'mango', 'tomato']
```

Appending with `extend(x)` method

In [158]:
```
fruit_basket
```

Out[158]:
```
['banana', 'orange', 'lime']
```

In [157]:
```
berries=['raspberry','strawberry']
```

In [159]:
```
fruit_basket.extend(berries) # append elements of a list to another li
st
print(fruit_basket)
```

```
['banana', 'orange', 'lime', 'raspberry', 'strawberry']
```

Appending a list to a list

In [161]:
```
veggies=['onion','garlic'] # append a list to a list
```

In [162]:
```
fruit_basket.append(veggies)
print(fruit_basket)
```

```
['banana', 'orange', 'lime', 'raspberry', 'strawberry', ['onion', 'g
arlic']]
```

In [163]:
```
fruit_basket[-1]
```

Out[163]:
```
['onion', 'garlic']
```

In [164]:
```
fruit_basket[-1][0]
```

Out[164]:
```
'onion'
```

Lists are mutable, while strings are immutable

In [165]:
```
fruit_basket[1]='nothing'
print(fruit_basket)
```

```
['banana', 'nothing', 'lime', 'raspberry', 'strawberry', ['onion', '
garlic']]
```

In [166]:
```
first_name
```

Out[166]:
```
'Nathalia'
```

In [170]:
```
first_name[2]='T'
```

```
---------------------------------------------------------------------
-------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-170-334a7e14fe01> in <module>
----> 1 first_name[2]='T'

TypeError: 'str' object does not support item assignment
```

LIST

Delete items from a list

In [171]:
```
del fruit_basket[-1]
```

In [172]:
```
print(fruit_basket)
```

```
['banana', 'nothing', 'lime', 'raspberry', 'strawberry']
```

In [173]:
```
fruit_basket.pop(1)
```

Out[173]:
```
'nothing'
```

In [174]:
```
fruit_basket
```

Out[174]:
```
['banana', 'lime', 'raspberry', 'strawberry']
```

In [175]:
```
fruit_basket.remove('lime')
```

In [176]:
```
fruit_basket
```

Out[176]:
```
['banana', 'raspberry', 'strawberry']
```

Sort - list function

In [177]:
```
ages
```

Out[177]:
```
[34, 40, 65]
```

In [178]:
```
ages.append(15)
print(ages)
```

```
[34, 40, 65, 15]
```

In [179]:
```
ages.append(40)
print(ages)
```

```
[34, 40, 65, 15, 40]
```

In [180]:
```
ages.append(a)
print(ages)
```

```
[34, 40, 65, 15, 40, 1.5]
```

In [181]:
```
print(sorted(ages))
```

```
[1.5, 15, 34, 40, 40, 65]
```

In [182]:
```
ages
```

Out[182]:
```
[34, 40, 65, 15, 40, 1.5]
```

In [183]:
```
sorted_ages=sorted(ages) # creates a copy of the original list
```

In [184]:
```
print(sorted_ages)
```

```
[1.5, 15, 34, 40, 40, 65]
```

In [185]:
```
ages
```

Out[185]:
```
[34, 40, 65, 15, 40, 1.5]
```

In [186]:
```
ages.sort() # sorts in place, changes the original
```

In [187]:
```
ages
```

Out[187]:
```
[1.5, 15, 34, 40, 40, 65]
```

Other list functions

In [188]:
```
#len(list)
#max(list)
#min(list)
```

FOR LOOPS

A for loop executes the same command through each value in a collection.

Building blocks of a for loop:

```
for each-item-in-this in variable :
(tab) do-something
```

Start the for loop with `for`
use `in` to indicate the variable
end the first line with `:`
indent the second line with `tab` (Jupyter does the indent automatically!)

`each-item-in-this` is an arbitrary name for each item in the variable/list.
`do-something` is a command.

In [189]:
```
fruit_basket
```

Out[189]:
```
['banana', 'raspberry', 'strawberry']
```

In [347]:
```
for number in range(10): # does not include 10!
    print(number)
```

```
0
1
2
3
4
5
6
7
8
9
```

In [348]:
```
total=0 # global variable

for i in range(10):
    total=total+i
    print(total)
```

```
0
1
3
6
10
15
21
28
36
45
```

In [349]:
```
total=0

for i in range(10):
    total=total+i

print(total)
```

```
45
```

In [350]:
```
data=[35,45,60,1.5,40,50]
```

In [351]:
```
for i in data:
    print(i*2)
```

```
70
90
120
3.0
80
100
```

`if/else` statements

conditional statement

serve to control the data inside the loop, control execution of the code

```
if some-condition :
(tab) do-something
else :
(tab) do-the-other-thing
```

Start with `if`
`some-condition` is a test that you want to run with your data
end the line with `:`
indent the line after if with `tab` (Jupyter does the indent automatically!)
End the statement with `else` and `:` (notice that if and else are in the same indent)
indent the line after the else with `tab`
`do-the-other-thing` is the opposite test from the if statement.

More Operators:

Comparison operators:

`==` equality

`!=` not equal

`>` greater than, `>=` greater than or equal to

`<` less than, `<=` less than or equal to

In [362]:
```
weight=3.56
if weight >= 2:
    print(weight,'is greater or equal than 2')
else:
    print(weight,'is less than 2')
```

```
3.56 is greater than 2
```

Membership operators:

in and not in

In [373]:
```
jewlery
```

Out[373]:
```
{'necklace': 3.45, 'bracelet': 70000.0}
```

In [388]:
```
if 'necklace' in jewlery:
    print('I added the necklace')
else:
    print('I forgot to add the necklace')
```

```
I added the price of the necklace
```

Logical operators:

not , or and and

In [395]:
```
if 'necklace' and 'bracelet' in jewlery:
    print('I added both')
else:
    print('I forgot to add one of them, or both')
```

```
I added both
```

In [397]:
```
if 'ring' or 'necklace' in jewlery:
    print('I added at least one of them')
else:
    print('I forgot to add one of them, or both')
```

```
I added both
```

In [400]:
```
if 'ring' and 'necklace' not in jewlery:
    print('I added at least one of them')
else:
    print('I forgot to add one of them, or both')
```

```
I forgot to add one of them, or both
```

Use `if/else` conditionals inside a loop

In [333]:
```
for i in data:
    if i %2 == 0:
        print(i, 'is even')
    else:
        print(i)
```

```
35
45
60 is even
1.5
40 is even
50 is even
```

In [337]:
```
print('Data that are even:')
for i in data:
    if i %2 == 0:
        print(i, end=' ')
    else:
        continue # do a print('a') to see what's going on...
```

```
Data that are even:
60 40 50
```

In [202]:
```
for i in range(15):
    if i <= 5:
        print(i)
        #continue
    else:
        print(i,'is greater than 5')
```

```
0
1
2
3
4
5
6 is greater than 5
7 is greater than 5
8 is greater than 5
9 is greater than 5
10 is greater than 5
11 is greater than 5
12 is greater than 5
13 is greater than 5
14 is greater than 5
```

More Operators:

Boolean operators:

`True` and `False`

They are also written by `False` == 0, and `True` ==1

In [405]:
```
fruit_basket
```

Out[405]:
```
['banana', 'raspberry', 'lime']
```

In [407]:
```
for fruit in fruit_basket:
    print(fruit.isdigit()) # test if it's a digit
```

```
False
False
False
```

In [409]:
```
for fruit in fruit_basket:
    if fruit.isdigit() == False:
        print(fruit)# test if it's a digit
```

```
banana
raspberry
lime
```

In [ ]:
```
for fruit in fruit_basket:
    if fruit.isdigit() == 0:
        print(fruit)# test if it's a digit
```

Challenge: reverse the string using a for loop

In [209]:
```
original = "waterfall"
result = ""
for char in original:
    result = char + result
#     print(char)
#     print(result)

print(result)
```

```
llafretaw
```

Challenge: perform some operation with values of a dictionary

In [472]:
```
data={'rainfall_inches':[1.34,1.56,4.33],'temperature_F':[75,80,96],'l
ocation':['A','B','C']}
```

In [474]:
```
for i in data:
    #print(i)
    #print(data[i])
    if 'rain' in i: # test if i startswith 'rain', if True goes inside
the elif loop; if False goes to else
        print('These are the rainfall in mm:')
        rain=data[i]
        for z in rain:
            print(round(z*25.4,1))

    elif i.startswith('temp'):
        print('These are the temperatures in Celsius:')
        temp=data[i]
        for j in temp:
            print(round((j-32)*5/9,2))

    else:
        continue
```

```
These are the rainfall in mm:
34.0
39.6
110.0
These are the temperatures in Celsius:
23.89
26.67
35.56
```

Functions:

Created your own functions, specially if you need to make the same operation many times. This will make you code cleaner.

In [166]:
```
def convert_temp(temperature,unit):
    """Function to convert temperature from F to C, and vice-versa.
    Need temperature (integer or float) and unit (string, uppercase F
or C)

    """
    t=int(temperature)
    u=str(unit)

    if u == 'C':
        fahr=(9/5*t)+32
        print( '{}C is {}F'.format(t,int(fahr)))

    elif u == 'F': # or else:
        celsius=(t-32)*5/9
        print('{}F is {}C'.format(t,int(celsius)))
```

In [87]:
```
convert_temp?
```

In [107]:
```
convert_temp(85,'C')
```

Out[107]:
```
'85C is 185F'
```

In [108]:
```
def convert_temp2():
    """Function to convert temperature from F to C, and vice-versa.
    Accept user input.

    """
    t=int(input('Enter temperature:'))
    u=str(input('Enter unit (F or C):'))

    if u == 'C':
        fahr=9/5*t+32
        return '{}C is {}F'.format(t,int(fahr))

    elif u == 'F':
        celsius=(t-32)*5/9
        return '{}F is {}C'.format(t,int(celsius))

    else:
        return "Don't know how to convert..."
```

In [109]:
```
convert_temp2()
```

```
Enter temperature:85
Enter unit (F or C):C
```

Out[109]:
```
'85C is 185F'
```

Getting closer to real life data analysis:

In [173]:
```
data_dict={}
data_dict['tempf']=[]
data_dict['tempc']=[]

with open('mock_data.txt','r') as data:
    next(data)
    for line in data:
        line=line.rstrip().split(',') # split data into a list of stri
ngs
        #print(line)
        temp_f=int(line[0])
        temp_c=int(line[1])
        data_dict['tempf'].append(temp_f)
        data_dict['tempc'].append(temp_c)


#data_dict
```

In [171]:
```
for temp in data_dict:
    if temp.endswith('c'):
        for v in data_dict[temp]:
            convert_temp(v,'C')
    else:
        for v in data_dict[temp]:
            convert_temp(v,'F')
```

```
85F is 29C
110F is 43C
34F is 1C
32C is 89F
15C is 59F
23C is 73F
```

In [180]:
```
# with glob

import glob

files=glob.glob('mock*data.txt')
# print(files) # list of file names that match that pattern

for file in files:
    data_dict={}
    data_dict['tempf']=[]
    data_dict['tempc']=[]
    with open(file,'r') as data:
        print(data.name)
        next(data)
        for line in data:
            line=line.rstrip().split(',') # split data into a list of
strings
            #print(line)
            temp_f=int(line[0])
            temp_c=int(line[1])
            data_dict['tempf'].append(temp_f)
            data_dict['tempc'].append(temp_c)

    for temp in data_dict:
        if temp.endswith('c'):
            for v in data_dict[temp]:
                convert_temp(v,'C')
        else:
            for v in data_dict[temp]:
                convert_temp(v,'F')
    print()
```

```
mock_data.txt
85F is 29C
110F is 43C
34F is 1C
32C is 89F
15C is 59F
23C is 73F

mock2_data.txt
85F is 29C
110F is 43C
34F is 1C
32C is 89F
15C is 59F
23C is 73F
```

Errors

Variable errors

In [210]:
```
# need to create/define a variable before using it
chocolate_cake
```

```
---------------------------------------------------------------------
-------
NameError                                 Traceback (most recent cal
l last)
<ipython-input-210-9507c04e8ac2> in <module>
      1 # need to create/define a variable before using it
----> 2 chocolate_cake

NameError: name 'chocolate_cake' is not defined
```

In [211]:
```
# this also includes mispellings...
firt_name
```

```
---------------------------------------------------------------------
-------
NameError                                 Traceback (most recent cal
l last)
<ipython-input-211-d0c0dec7f1dd> in <module>
      1 # this also includes mispellings...
----> 2 firt_name

NameError: name 'firt_name' is not defined
```

In [212]:
```
first_name
```

Out[212]:
```
'Nathalia'
```

Syntax errors

In [410]:
```
# Syntax errors: when you forget to close a )
## EOF - end of file
## means that the end of your source code was reached before all code blocks were completed
print(len(first_name)
```

```
  File "<ipython-input-410-9aa910f8efe1>", line 4
    print(len(first_name)
                        ^
SyntaxError: unexpected EOF while parsing
```

In [411]:
```
print(len(first_name))
```

```
8
```

In [412]:
```
# Syntax errors: when you forgot a ,
print(first_name,'age is:'age)
```

```
  File "<ipython-input-412-91603502cda3>", line 2
    print(first_name,'age is:'age)
                             ^
SyntaxError: invalid syntax
```

In [413]:
```
print(first_name,'age is:',age)
```

```
Nathalia age is: 31
```

In [414]:
```
# Syntax errors: forgot to close a quote ' in a string
## EOL = end of line
print(first_name,'age is:,age)
```

```
  File "<ipython-input-414-4fcc5cd9871d>", line 3
    print(first_name,'age is:,age)
                              ^
SyntaxError: EOL while scanning string literal
```

In [415]:
```
print(first_name,'age is:',age)
```

```
Nathalia age is: 31
```

In [416]:
```
# Syntax errors: when you forget the colon at the end of a line
for i in data
    print(i**2)
```

```
  File "<ipython-input-416-ec20c8c9a5f0>", line 2
    for i in data
                 ^
SyntaxError: invalid syntax
```

In [417]:
```
for i in data:
    print(i**2)
```

```
1225
2025
3600
2.25
1600
2500
```

In [418]:
```
# Indentation errors: forgot to indent
for i in data:
print(i**2)
```

```
  File "<ipython-input-418-9354b20bbb57>", line 3
    print(i**2)
        ^
IndentationError: expected an indented block
```

In [419]:
```
for i in data:
    print(i**2)
```

```
1225
2025
3600
2.25
1600
2500
```

BUT character strings are IMMUTABLE

In [420]:
```
fruit
```

Out[420]: 'lime'

In [421]:
```
fruit[3]
```

Out[421]: 'e'

In [422]:
```
fruit[3]='K'
```

```
---------------------------------------------------------------------
-------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-422-1f67ea88cf04> in <module>
----> 1 fruit[3]='K'

TypeError: 'str' object does not support item assignment
```

In [423]:
```
fruit
```

Out[423]: 'lime'

In [424]:
```
fruit_basket
```

Out[424]: ['banana', 'raspberry', 'lime']

In [425]:
```
fruit_basket[2]
```

Out[425]: 'lime'

In [426]:
```
fruit_basket[2][2]
```

Out[426]: 'm'

In [427]:
```
fruit_basket[2][2]='K'
```

```
---------------------------------------------------------------------
-------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-427-6555f952ef44> in <module>
----> 1 fruit_basket[2][2]='K'

TypeError: 'str' object does not support item assignment
```

In [430]:
```
fruit_basket[2]='lemon'
```

In [431]:
```
fruit_basket
```

Out[431]:
```
['banana', 'raspberry', 'lemon']
```

Index errors

In [434]:
```
# Index error: when caling an index that does not exist
fruit_basket[5]
```

```
---------------------------------------------------------------------
-------
IndexError                                Traceback (most recent cal
l last)
<ipython-input-434-d4ec8690e260> in <module>
      1 # Index error: when caling an index that does not exist
----> 2 fruit_basket[5]

IndexError: list index out of range
```

In [435]:
```
fruit_basket[:]
```

Out[435]:
```
['banana', 'raspberry', 'lemon']
```

Help yo'self (:

In [237]:
```
help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=Fals
e)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.
stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

In [238]:
```
help(len)
```

```
Help on built-in function len in module builtins:

len(obj, /)
    Return the number of items in a container.
```

```
help(your_data_object)
dir(your_data_object)
```

Items with double underscores have to do with classes, another type of object which is beyond the scope of this lesson to discuss. So for now, only focus your attention to the names that do not start with double underscores.

In [ ]:
```

```