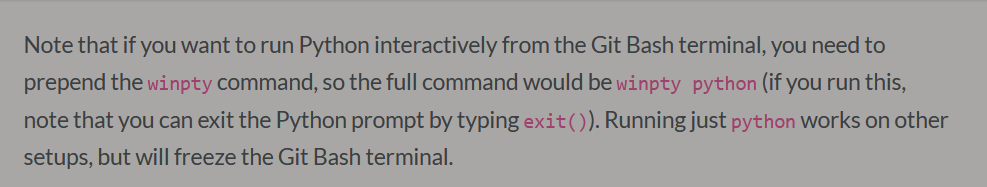
Python Beginner Notes

* Datatypes –
  + Value
    - Piece of data that a computer program works with such as a number or text
    - E.g 42 (integer), “hello” (string)
  + Variable
    - Name that refers to a value
      * E.g. x and y
      * In Python, any word can be used as a variable (exceptions: reserved words, such as: for, while, class, lambda, etc.)
    - Assignment operator—to assign a value to a variable

**Common built-in data types**

* Floating point number (float) – real number in decimal form
* Boolean (bool) – true or false
* String (str) – text (e.g. “all your base are belong to us”)
* List (list) – homogeneous sequence representing a collection of similar objects
  + E.g. ‘Ali’, ‘Miriam’, ‘Kelly’)
* Tuple (tuple) – heterogeneous sequence representing a single object
  + E.g. ‘Thursday’, 6,9,2018)
* Dictionary (dict) –mapping of key-value pairs (e.g. ‘name’ : ‘DSCI’ , ‘code’ :511, ‘credits’ :2)
* None (NoneType) – represents no value (e.g. None)



Numeric Types

* X= 14
  + Type(x)
  + Int  
    - Print(x)
      * 14
      * \*don’t always have to type print() in Jupyter for variables
* Arithmetic operators
  + \*\* exponentials
  + // integer division
  + % modulo – remainder when odd number is divided using //
* None – nonetype has one possible value: none
  + E.g. x=none
    - Print(x): none
* Strings – text is stored as such
* Boolean—2 values – True and False
  + E.g. the\_truth = True
    - Print(the\_truth): True
* Comparison Operators
  + Compares objects using comparison operators. Result is Boolean values
  + Notables: x==y – is x equal to y?
    - x!=y—is x not equal to y?
    - 2>3: True
    - 2==”2”: False
    - 2==2.0: True
  + Operators on Boolean values
    - X and y: are both x and y true?
    - X or y: is one of x or y true?
    - Not x : is x false?

Casting

* Changes a value from one type to another
  + E.g. x= int(5.0): 5
    - X= str(5.0): ‘5.0’  
      type(x): str

Lists and Tuples

* Allow us to store multiple things (“elements”) in a single object
  + Elements are ordered
  + E.g. my\_list = [1, 2, “Three”, 4, 0.5]
    - Print(my\_list): [1, 2, ‘Three’, 4, 0.5]
    - Type(my\_list): list
  + Get length of list by typing: len(my\_list)
* Indexing and splicing sequences
  + List a specific value from a list
    - E.g. my\_list: [1, 2, “Three”, 4, 0.5]
    - My\_list[0]: 1
    - My\_list[4]: 0.5
  + Negative indices cand be used to count backwards from the end of the list
    - E.g. my\_list[-1]: 0.5
  + Colons used to list a sequence of numbers (like excel ranges)
    - E.g. my\_list[1:4]: [2, ‘Three’, 4]—note: end number is exclusive (did not list value#4)
    - Start or end can be omitted: my\_list[:3]: [1, 2, ‘Three’]
      * My\_list[3:]: [4, 0.5]
    - My\_list[:] : [1, 2, ‘Three’, 4, 0.5]

List Methods

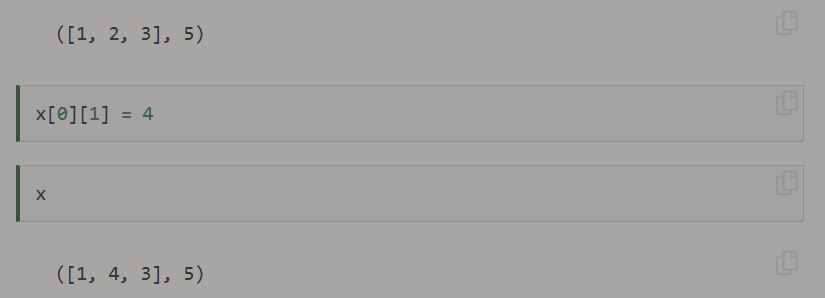
* A list is an object and it has methods for interacting with its data
  + E.g. list.append(item) – appends an item to the end of the list

Sets

* Set – stores and un-ordered list of unique items
  + E.g. s= {2,3, 5, 11}
    - S: {2, 3, 5, 11}
  + {1,2,3] == {3,2,1} : True
  + [1,2,3] == [3,2,1] : False

Mutable vs immutable types

* Strings and tuples are immutable types (cannot be modified)
* Lists are mutable – we can assign new values for its various entries
  + Main difference between lists and tuples
* E.g. names\_list = [“Indiana”, “Fang”, “Linsey”]
  + Names\_list: [“Indiana”, “Fang”, “Linsey”]
    - Names\_list[0] = “Cool guy”
    - Names\_list : [“Cool guy”, “Fang”, “Linsey”]
    - Names\_tuple = [“Indiana”, “Fang”, “Linsey”]
      * Names\_tuple: [“Indiana”, “Fang”, “Linsey”]
      * Trying to modify the tuple leads to an error
* Strings can therefore not be redefined

**However**: 

* String methods

: all\_caps = “HOW ARE YOU TODAY?”

* + - Print(all\_caps) : HOW ARE YOU TODAY?
    - New\_str = all\_caps.lower()
    - New\_str: ‘how are you today?’
      * Doesn’t change original; instead, it just returns a new string
* String formatting
  + E.g. template\_new = “Hello, my name is {}. I am {:.2f} years old.”
    - Template\_new.format(‘Newborn baby’, 4/12)  
      * ‘Hello, my name is Newborn Baby. I am 0.33 years old.’

OR  
  
name= “Newborn Baby”

Age= 4/12

Template\_new= f’Hello, my name is {name}. I am {age: .2f} years old.’  
template\_new

‘Hello, my name is Newborn Baby. I am 0.33 years old.’

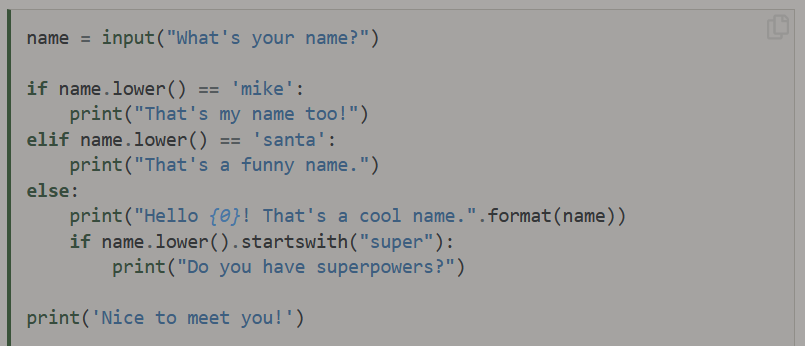
Dictionaries

* Mapping between key-values pairs
  + E.g. house = {‘bedrooms’: 3, ‘bathrooms’: 2, ‘city’: ‘Vancouver’, ‘price’: 2499999, ‘date\_sold’: (1,3,2015)}
  + Condo = {‘bedrooms’ : 2,   
    ‘bathrooms’ :1,  
    ‘city’ : ‘Burnaby’  
    ‘price’ : 699999  
    ‘date\_sold’: (27,8,2011)

}

* + We can now access a specific field of a dictionary with square brackets:
    - House[‘price] : 2499999
    - Condo[‘city’] : ‘Burnaby’
* We can also edit dictionaries: e.g. cond[‘price’] = 5 – changes price to 5
  + Condo[‘flooring’] = “wood” –adds line to condo
  + Fields can also be deleted—e.g. del condo[“city”] – deletes ‘city’ info
  + Can also add existing fields –e.g. condo[5] = 443345 – adds to description (5 refers to line 5, which is empty)
* Useful tricks – condo[“bedrooms”] : 2
  + Shorthand for: condo.get(“bedrooms”)
  + Condo.get(“fireplaces”, “unknown”)
    - : ‘unknown’
  + Maximum dictionary key by value: max(word\_lengths, key=word\_lengths.get)
    - Leads to error
* Empties
  + E.g. 1st=list() – empty list
    - : []
  + Tup= tuple() – empty tuple
    - : ()
    - Or tup= ()
  + Dic = dict()
    - Dic: {}
    - St= set()
      * : set()
    - St = {} – NOT AN EMPTY SET
      * : type(st): dict
* Conditionals
  + Allow us to write programs where only certain blocks of code are executed depending on the state of the program
  + Name = input (“What’s your name?”)
  + If name.lower() == ‘mike’:   
     print(“That’s my name too!”)  
    elif name.lower() == ‘santa’:   
     print(“That’s a funny name.”)  
    else: print(“Hello {}! That’s a cool name.” .format(name))

print(‘Nice to meet you!’)

* + - E.g. What’s your name?mike  
      That’s my name too!
      * Colon ends conditional expression
      * Indentation (4 empty spaces) defines code blocks
      * In an if statement, the first block whose conditional statement returns True is executed and the program exists the if block
      * If statements don’t necessarily need elif or else
      * Elif—leets us check several conditions
      * Else—lets us evaluate a default black if all other conditions are FALSE
      * The end of the entire if statement is where the indentation returns to the same level as the first if keyword
    - If statements can also be nested inside of one another:
    - 
    - E.g. What’s your name?supersam

Hello supersam! That’s a cool name.

Do you have superpowers?

Nice to meet you!

Inline if/else

Words= [“the”, “list”, “of”, “words”]

X= “long list” if len(words) > 10 else “short list”

X : ‘short list’

If len(words) > 10:

X= “long list”

Else:

X= “short list”

X: ‘short list’

* Short-circuiting : BLAH – not defined – can be used to replace either true or false
  + E.g. True or BLAH
  + E.g. False and BLAH