Week I

Varibles and Assignments

```
#assignment of variables
name = "Alfredo"
last_name = "Deza"
```

```
#no need to worry about type of value
height = 1000
distance = 1.33
date = "Tuesday"
```

```
#lack of type checking can make things tricky
height = "10000"
height
```

```
'10000'
```

```
#re-assigning variables
name = "James"
name
```

```
'James'
```

```
#using print() with variables
print(name, last_name)
```

```
James Deza
```

```
#Creating new variables from existing variables
#watch out with strings
#full_name = name +last_name
full_name = f"{name} {last_name}"
full_name
```

```
'James Deza'
```

```
#Watch out for copying variables
new_name = name
print(new_name)
name = "Alfredo"
print(new_name)
print(name)
```

```
James
James
Alfredo
```

```
#the print()function helps to display the values
#it adds separation when using commas
print(name, last_name, "is your instructor", "today.")
```

```
Alfredo Deza is your instructor today.
```

Working with Different Data Types

```
#strings can be assigned with single, double, and triple quotes
full_name = 'Alfredo Deza '
```

```
full_name = "Alfredo Deza "
```

```
full_name = """Alfredo Deza """
```

```
#Triple quotes are useful when having quotes within a string
summary = """Triple quotes are useful when you have a 'or a  "within a string"""
print(summary)
```

```
Triple quotes are useful when you have a 'or a  "within a string
```

```
#use single quotes when u habe a double quotes in your string
summary = 'use single quotes when u habe a double " in your string'
print(summary)
print("And use double quote when u have ' in your string")
```

```
use single quotes when u habe a double " in your string
And use double quote when u have ' in your string
```

```
name = " Alfredo "
result = "this result is brought to u by" + name
print(result)
```

```
this result is brought to u by Alfredo
```

```
#use f string to replace variable in a string
result = f"{name} is teaching f strings!"
print(result)
```

```
 Alfredo  is teaching f strings!
```

Integers

```
#use type() to discover what type it is if u don't know
type(15)
```

```
int
```

```
#integers support mathematical operations
14/2
```

```
7.0
```

```
#watchout for invalid mathematical operations
7/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-18-ff4b999c47a3> in <cell line: 1>()
----> 1 7/0

ZeroDivisionError: division by zero
```

```
#as with all other types, watch out when mixing unsupported types
7+ "14"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-19-f5ed5078d900> in <cell line: 2>()
      1 #as with all other types, watch out when mixing unsupported types
----> 2 7+ "14"

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

```
#type can change depennding upon operations
result = 3/2
print(result)
type(result)
```

```
1.5
float
```

Floats

```
type(14.3)
```

```
float
```

```
311/99
```

```
3.1414141414141414
```

Booleans

```
type(True)
```

```
bool
```

```
#truthy values can be converted to booleans with the bool() built-in
first_result = bool(1)
second_result = bool(0)
print(first_result, second_result)
```

```
True False
```

none

```
type(None)
```

```
NoneType
```

Conditionals and Evaluations

```
#all conditions evaluate the result of condition(expression)
condition = True
if condition:
  print("the condition was met")
```

```
the condition was met
```

```
#if the condition gets false,it gets skipped
condition = False
if condition:
  print("Was the condition met?")
```

```
#Some data Structure is truthy: false when empty, but true when they contain items
groceries = []
if groceries:
  print("We have some groceries")

invites = ["john", "George"]
if invites:
  print("We have some invites!")
```

```
We have some invites!
```

```
#other types like integers(0, and any positive integer ) are truthy
properties = 0
if properties:
  print("We have properties!")

parents = 2
if parents:
  print("WE have parents!")
```

```
WE have parents!
```

```
#operators are supported
if properties == 0:
  print("We have no properties!")

if parents>1:
  print("We have more than 1 parents")
```

```
We have no properties!
We have more than 1 parents
```

else conditions

```
if properties:
  print("We have properties!")
else:
  print("We don't have properties!")
```

```
We don't have properties!
```

elif conditions

```
if properties:
  print("We have properties")
  #evaluate if parents is valid
elif parents:
  print("We don't have any properties, but we have parents")
```

```
We don't have any properties, but we have parents
```

Negative Conditions

```
name = None
if not name:
  print("Didn't get a name")
```

```
Didn't get a name
```

```
#same is possible with elif condition
name = "Alfredo"
last_name = None
if not name:
  print("No name!")
elif not last_name:
  print("No last name either!")
```

```
No last name either!
```

Compounding Conditions with and

```
has_kids = True
married = True
if has_kids and married:
  print("This person is married and has kids")
```

```
This person is married and has kids
```

```
# not can be used but can get harder to read
likes_books = False
is_logged_in = False
if not likes_books and not is_logged_in:
  print("User not logged in!")
```

```
User not logged in!
```

Exceptions in Python

```
raise RuntimeError("this is a problm!")
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-14-f9d45859d2be> in <cell line: 1>()
----> 1 raise RuntimeError("this is a problm!")

RuntimeError: this is a problm!
```

```
try:
  #some intense operation that cause an error
  result = 14/0
except ZeroDivisionError :
  #do some other intense operation
  result = 14/2
  print(result)
```

```
7.0
```

Catching Multiple Exceptions

```
try:
  #some intense operation that cause an error
  result = 14/0
  result + "100"
except ( ZeroDivisionError, TypeError) :
  #do some other intense operation
  result = 14/2
  print(result)
```

```
7.0
```

```
#assign the resulting exception to a variable
try:
  #some intense operation that cause an error
  result = 14/0
except ZeroDivisionError as error:
  #do some other intense operation
  print(f"got an error --> {error}")
  result = 14/2
  print(result)
```

```
got an error --> division by zero
7.0
```

Python Data Structure

Introduction to list

```
#list are container of item identified by square bracket
[1, 2, 3, 4, 5]
```

```
[1, 2, 3, 4, 5]
```

```
#list can contain different types of item in them
[1, "two", False, 4.5]
```

```
[1, 'two', False, 4.5]
```

```
# use built-in to count items in a list
len([3, 4, "red", "car"])
```

```
4
```

```
#each item has a position called index.Count starts with 0
items = ["carrot", "peas", "celery"]
#retreiving an item is done with index
items[0]
```

```
'carrot'
```

```
#retreive last item
items[-1]
```

```
'celery'
```

```
#negative count from the last
items[-2]
```

```
'peas'
```

creating list

```
#create one by defining empty list with brackets
items = []
items
```

```
[]
```

```
#use the built-in list in python is more common to use brackets
items = list()
items
```

```
[]
```

```
# data can be pre-seeded
colors = ["red","blue","brown"]
colors
```

```
['red', 'blue', 'brown']
```

Iterating over list

```
for color in colors:
    print(color)
```

```
red
blue
brown
```

```
for color in ["yellow", "red"]:
  print(color)
```

```
yellow
red
```

List Comprehensions

```
numbers = [2, 3, 4, 12, 5, 3, 4]
low_numbers = [n for n in numbers if n>6]
low_numbers
```

```
[12]
```

Introduction to Dictionaries

```
#curly brackets are required to create one
{}
```

```
{}
```

```
#you always map a key to a value
{"key":"value"}
```

```
{'key': 'value'}
```

```
#the value can be other type
{"key": True}
```

```
{'key': True}
```

```
#but the key has to be unique ,there can't be duplicates
{"name":"Alfredo", "name":"Alfredo"}
```

```
{'name': 'Alfredo'}
```

```
#value can be other dictionaries or list
{"items":["lumber","concerete","nails"]}
```

```
{'items': ['lumber', 'concerete', 'nails']}
```

```
#you can't have the list or a dict as a key
{[1,2]:False}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-22-8673f0749d81> in <cell line: 2>()
      1 #you can't have the list or a dict as a key
----> 2 {[1,2]:False}

TypeError: unhashable type: 'list'
```

```
#with curly bracket
contact_information = {}
contact_information
```

```
{}
```

```
#with curly bracket
contact_information = {"name":"Alfredo","Last_name":"Deza"}
contact_information
```

```
{'name': 'Alfredo', 'Last_name': 'Deza'}
```

```
# with the dict built-in
contact_information = dict()
contact_information
```

```
{}
```

```
#with dict() and a list of tuple pairs
data =[("name","Alfredo"),("Last_name","Deza")]
dict(data)
```

```
{'name': 'Alfredo', 'Last_name': 'Deza'}
```

```
# with dict() and keyword  arguement
dict(first="alfredo",lastname="deza")
```

```
{'first': 'alfredo', 'lastname': 'deza'}
```

Looping over Dictionaries

```
contact_imformation = {"name":"Alfredo",
                       "last_name" : "Deza",
                       "height" :112.4,
                       "age":49}
```

```
#by default
# for key in contact_imformation:
#   print(key)

#explicit
for key in contact_imformation.keys():
  print(key)
```

```
name
last_name
height
age
```

```
# retrieve only values
for value in contact_imformation.values():
  print(value)
```

```
Alfredo
Deza
112.4
49
```

```
# retrieve both keys and values
for key, value in contact_imformation.items():
  print(f"{key}-->{value}")
contact_imformation.items()
```

```
name-->Alfredo
last_name-->Deza
height-->112.4
age-->49
dict_items([('name', 'Alfredo'), ('last_name', 'Deza'), ('height', 112.4), ('age', 49)])
```

Tuple

```
ro_items = ('first','second','third')
print(ro_items[-1])
print("first item in the tuple is:%s"%ro_items.index('first'))
for item in ro_items:
  print(item)
```

```
third
first item in the tuple is:0
first
second
third
```

```
#expect an error
ro_items[9]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-7-aa0f7d380806> in <cell line: 2>()
      1 #expect an error
----> 2 ro_items[9]

IndexError: tuple index out of range
```

```
#same with indexes
ro_items.index('fifth')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-9-244ae8b3b1f5> in <cell line: 2>()
      1 #same with indexes
----> 2 ro_items.index('fifth')

ValueError: tuple.index(x): x not in tuple
```

```
# find out what methods are available in a tuple
for method in dir(tuple):
  if method.startswith('_'):
    continue
  print(method)
```

```
count
index
```

```
# tuples are immutable
ro_items.append('a')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-11-00097ba767e9> in <cell line: 2>()
      1 # tuples are immutable
----> 2 ro_items.append('a')

AttributeError: 'tuple' object has no attribute 'append'
```

Sets

```
#crwating an empty sets
unique = set()
#add items with .add()
unique.add("one")
unique
```

```
{'one'}
```

```
# adding more items as long as they are unique
unique.add("one")
unique.add("one")
unique.add("one")
unique.add("two")
unique
```

```
{'one', 'two'}
```

```
# you can pop items in like list but it takes no arguement
unique.pop()
unique
```

```
{'one'}
```

List Comprehensions

```
items = ['a', '1', '23', 'b', '4', 'c', 'd']
numeric = []
for item in items:
  if item.isnumeric():
    numeric.append(item)
print(numeric)
```

```
['1', '23', '4']
```

```
# notice the `if` condition at the end, is this more readable? or less?
inlined_numeric = [item for item in items if item.isnumeric()]
inlined_numeric
```

```
['1', '23', '4']
```

```
# doubly nested items are usually targetted for list comprehensions
items = ['a', '1', '23', 'b', '4', 'c', 'd']
nested_items = [items, items]
nested_items
```

```
[['a', '1', '23', 'b', '4', 'c', 'd'], ['a', '1', '23', 'b', '4', 'c', 'd']]
```

```
numeric = []
for parent in nested_items:
    for item in parent:
      if item.isnumeric():
        numeric.append(item)
numeric
```

```
['1', '23', '4', '1', '23', '4']
```

```
# and now with list comprehensions
numeric = [item for item in parent for parent in nested_items if item.isnumeric()]
numeric
```

```
['1', '1', '23', '23', '4', '4']
```

```
# this can improve readability
numeric = [
    item for item in parent
        for parent in nested_items
            if item.isnumeric()
]
numeric
```

```
['1', '1', '23', '23', '4', '4']
```

```
# dictionaries are mappings, usually referred to as key/value mappings
contacts = {
    'alfredo': '+3 678-677-0000',
    'noah': '+3 707-777-9191'
}
contacts
```

```
{'alfredo': '+3 678-677-0000', 'noah': '+3 707-777-9191'}
```

```
contacts['noah']
```

```
'+3 707-777-9191'
```

```
# you can get keys as list-like objects
contacts.keys()
```

```
dict_keys(['alfredo', 'noah'])
```

```
# or you can get the values as well
contacts.values()
```

```
dict_values(['+3 678-677-0000', '+3 707-777-9191'])
```

```
# looping over dictionaries default to `.keys()` and you can loop over both keys and values
for key in contacts:
  print(key)
for name, phone in contacts.items():
  print("Key: {0}, Value: {1}".format(name, phone))
```

```
alfredo
noah
Key: alfredo, Value: +3 678-677-0000
Key: noah, Value: +3 707-777-9191
```

Adding Data to lists

```
# define an empty list of fruits
fruits = []
fruits
```

```
[]
```

```
# appending items will insert them by the end
fruits.append("Orange")
fruits.append("apple")
fruits
```

```
['Orange', 'apple']
```

```
# similar to append u can insert but that require an index position
fruits.insert(0, "melon")
fruits
```

```
['melon', 'Orange', 'apple']
```

```
# you can add one list to another
vegetables = ['cucumber','carrot']
fruits+ vegetables
```

```
['melon', 'Orange', 'apple', 'cucumber', 'carrot']
```

```
# watch out when appending a list to an existing list
shopping_list = fruits+vegetables
shopping_list.append(["sugar","salt"])
shopping_list
```

```
['melon', 'Orange', 'apple', 'cucumber', 'carrot', ['sugar', 'salt']]
```

```
# you can extend a list instead(similar to adding)
shopping_list = fruits+vegetables
shopping_list.extend(["sugar","salt"])
shopping_list
```

```
['melon', 'Orange', 'apple', 'cucumber', 'carrot', 'sugar', 'salt']
```

Extracting Data from List

```
colors = ["red","yellow","green","blue"]
# by index
colors[0]
```

```
'red'
```

```
# slicing for the first three items
colors[:2]
```

```
['red', 'yellow']
```

```
# slicing for the last three items
colors[-3:]
```

```
['yellow', 'green', 'blue']
```

```
# slicing for a range
colors[1:3]
```

```
['yellow', 'green']
```

```
# popping can retreive an item and removing it from the list
# but the index must exist
colors.pop(100)
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-12-97ca55bfb519> in <cell line: 3>()
      1 # popping can retreive an item and removing it from the list
      2 # but the index must exist
----> 3 colors.pop(100)

IndexError: pop index out of range
```

```
# use an existing index to pop properly
popped_item = colors.pop(1)
popped_item
```

```
'yellow'
```

```
# popping alters the list however
colors
```

```
['red', 'green', 'blue']
['red', 'green', 'blue']
```

```
colors.remove("blue")
colors
```

```
['red', 'green']
```

Retreiving Data

```
contact_imformation = {}
```

```
# normal retreival[and possible exception]
contact_imformation["height"]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-29-f812c7587883> in <cell line: 2>()
      1 # normal retreival[and possible exception]
----> 2 contact_imformation["height"]

KeyError: 'height'
```

```
# using a try/except block
try:
  contact_imformation["height"]
except KeyError:
  print("6ft")
```

```
6ft
```

```
# using .get()
result = contact_imformation.get("height")
print("Height of contact is", result)
```

```
Height of contact is None
```

```
# falling back when there is no key
result = contact_imformation.get("height", "5  ft 9 in"  )
print("Height of contact is", result)
```

```
Height of contact is 5  ft 9 in
```

```
contact_imformation["age"] = 31
print("Age is", conatct_imformation.pop("age"))
print(contact_imformation)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-33-3f52a658e769> in <cell line: 2>()
      1 contact_imformation["age"] = 31
----> 2 print("Age is", conatct_imformation.pop("age"))
      3 print(contact_imformation)

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