

# IntroductionToCourse

March 1, 2021

---

*You are currently looking at **version 1.1** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the [Jupyter Notebook FAQ](#) course resource.*

---

## 1 The Python Programming Language: Functions

```
[1]: x = 1
     y = 2
     x + y
```

```
[1]: 3
```

```
[2]: y
```

```
[2]: 2
```

`add_numbers` is a function that takes two numbers and adds them together.

```
[3]: def add_numbers(x, y):
     return x + y

     add_numbers(1, 2)
```

```
[3]: 3
```

'`add_numbers`' updated to take an optional 3rd parameter. Using `print` allows printing of multiple expressions within a single cell.

```
[4]: def add_numbers(x, y, z=None):
     if (z==None):
         return x+y
     else:
         return x+y+z

     print(add_numbers(1, 2))
     print(add_numbers(1, 2, 3))
```

```
3
6
```

add\_numbers updated to take an optional flag parameter.

```
[5]: def add_numbers(x, y, z=None, flag=False):  
    if (flag):  
        print('Flag is true!')  
    if (z==None):  
        return x + y  
    else:  
        return x + y + z  
  
print(add_numbers(1, 2, flag=True))
```

Flag is true!

3

Assign function add\_numbers to variable a.

```
[6]: def add_numbers(x,y):  
    return x+y  
  
a = add_numbers  
a(1,2)
```

[6]: 3

## 2 The Python Programming Language: Types and Sequences

Use type to return the object's type.

```
[7]: type('This is a string')
```

[7]: str

```
[8]: type(None)
```

[8]: NoneType

```
[9]: type(1)
```

[9]: int

```
[10]: type(1.0)
```

[10]: float

```
[11]: type(add_numbers)
```

[11]: function

Tuples are an immutable data structure (cannot be altered).

```
[12]: x = (1, 'a', 2, 'b')  
type(x)
```

[12]: tuple

Lists are a mutable data structure.

```
[13]: x = [1, 'a', 2, 'b']  
      type(x)
```

[13]: list

Use append to append an object to a list.

```
[14]: x.append(3.3)  
      print(x)
```

[1, 'a', 2, 'b', 3.3]

This is an example of how to loop through each item in the list.

```
[15]: for item in x:  
      print(item)
```

1  
a  
2  
b  
3.3

Or using the indexing operator:

```
[16]: i=0  
      while( i != len(x) ):  
          print(x[i])  
          i = i + 1
```

1  
a  
2  
b  
3.3

Use + to concatenate lists.

```
[17]: [1,2] + [3,4]
```

[17]: [1, 2, 3, 4]

Use \* to repeat lists.

```
[18]: [1]*3
```

[18]: [1, 1, 1]

Use the in operator to check if something is inside a list.

```
[19]: 1 in [1, 2, 3]
```

[19]: True

Now let's look at strings. Use bracket notation to slice a string.

```
[20]: x = 'This is a string'
print(x[0]) #first character
print(x[0:1]) #first character, but we have explicitly set the end character
print(x[0:2]) #first two characters
```

T  
T  
Th

This will return the last element of the string.

```
[21]: x[-1]
```

```
[21]: 'g'
```

This will return the slice starting from the 4th element from the end and stopping before the 2nd element from the end.

```
[22]: x[-4:-2]
```

```
[22]: 'ri'
```

This is a slice from the beginning of the string and stopping before the 3rd element.

```
[23]: x[:3]
```

```
[23]: 'Thi'
```

And this is a slice starting from the 4th element of the string and going all the way to the end.

```
[24]: x[3:]
```

```
[24]: 's is a string'
```

```
[25]: firstname = 'Christopher'
      lastname = 'Brooks'

print(firstname + ' ' + lastname)
print(firstname*3)
print('Chris' in firstname)
```

Christopher Brooks  
ChristopherChristopherChristopher  
True

split returns a list of all the words in a string, or a list split on a specific character.

```
[26]: firstname = 'Christopher Arthur Hansen Brooks'.split(' ')[0] # [0] selects the
      →first element of the list
      lastname = 'Christopher Arthur Hansen Brooks'.split(' ')[-1] # [-1] selects the
      →last element of the list
print(firstname)
print(lastname)
```

Christopher  
Brooks

Make sure you convert objects to strings before concatenating.

```
[27]: 'Chris' + 2
```

```
TypeError                                Traceback (most recent call
↳last)

<ipython-input-27-9d01956b24db> in <module>
----> 1 'Chris' + 2

TypeError: can only concatenate str (not "int") to str
```

```
[ ]: 'Chris' + str(2)
```

Dictionaries associate keys with values.

```
[ ]: x = {'Christopher Brooks': 'brooks@umich.edu', 'Bill Gates': 'billg@microsoft.com'}
      x['Christopher Brooks'] # Retrieve a value by using the indexing operator
```

```
[ ]: x['Kevyn Collins-Thompson'] = None
      x['Kevyn Collins-Thompson']
```

Iterate over all of the keys:

```
[ ]: for name in x:
      print(x[name])
```

Iterate over all of the values:

```
[ ]: for email in x.values():
      print(email)
```

Iterate over all of the items in the list:

```
[ ]: for name, email in x.items():
      print(name)
      print(email)
```

You can unpack a sequence into different variables:

```
[ ]: x = ('Christopher', 'Brooks', 'brooks@umich.edu')
      fname, lname, email = x
```

```
[ ]: fname
```

```
[ ]: lname
```

Make sure the number of values you are unpacking matches the number of variables being assigned.

```
[ ]: x = ('Christopher', 'Brooks', 'brooks@umich.edu', 'Ann Arbor')
      fname, lname, email = x
```

### 3 The Python Programming Language: More on Strings

```
[ ]: print('Chris' + 2)
```

```
[ ]: print('Chris' + str(2))
```

Python has a built in method for convenient string formatting.

```
[ ]: sales_record = {
      'price': 3.24,
      'num_items': 4,
      'person': 'Chris'}

      sales_statement = '{} bought {} item(s) at a price of {} each for a total of {}'

      print(sales_statement.format(sales_record['person'],
                                  sales_record['num_items'],
                                  sales_record['price'],
                                  sales_record['num_items']*sales_record['price']))
```

### 4 Reading and Writing CSV files

Let's import our datafile mpg.csv, which contains fuel economy data for 234 cars.

- mpg : miles per gallon
- class : car classification
- cty : city mpg
- cyl : # of cylinders
- displ : engine displacement in liters
- drv : f = front-wheel drive, r = rear wheel drive, 4 = 4wd
- fl : fuel (e = ethanol E85, d = diesel, r = regular, p = premium, c = CNG)
- hwy : highway mpg
- manufacturer : automobile manufacturer
- model : model of car
- trans : type of transmission
- year : model year

```
[ ]: import csv

      %precision 2

      with open('mpg.csv') as csvfile:
          mpg = list(csv.DictReader(csvfile))
```

```
mpg[:3] # The first three dictionaries in our list.
```

csv.Dictreader has read in each row of our csv file as a dictionary. len shows that our list is comprised of 234 dictionaries.

```
[ ]: len(mpg)
```

keys gives us the column names of our csv.

```
[ ]: mpg[0].keys()
```

This is how to find the average cty fuel economy across all cars. All values in the dictionaries are strings, so we need to convert to float.

```
[ ]: sum(float(d['cty']) for d in mpg) / len(mpg)
```

Similarly this is how to find the average hwy fuel economy across all cars.

```
[ ]: sum(float(d['hwy']) for d in mpg) / len(mpg)
```

Use set to return the unique values for the number of cylinders the cars in our dataset have.

```
[ ]: cylinders = set(d['cyl'] for d in mpg)
cylinders
```

Here's a more complex example where we are grouping the cars by number of cylinder, and finding the average cty mpg for each group.

```
[ ]: CtyMpgByCyl = []

for c in cylinders: # iterate over all the cylinder levels
    summpg = 0
    cyltypecount = 0
    for d in mpg: # iterate over all dictionaries
        if d['cyl'] == c: # if the cylinder level type matches,
            summpg += float(d['cty']) # add the cty mpg
            cyltypecount += 1 # increment the count
    CtyMpgByCyl.append((c, summpg / cyltypecount)) # append the tuple
    →('cylinder', 'avg mpg')

CtyMpgByCyl.sort(key=lambda x: x[0])
CtyMpgByCyl
```

Use set to return the unique values for the class types in our dataset.

```
[ ]: vehicleclass = set(d['class'] for d in mpg) # what are the class types
vehicleclass
```

And here's an example of how to find the average hwy mpg for each class of vehicle in our dataset.

```
[ ]: HwyMpgByClass = []

for t in vehicleclass: # iterate over all the vehicle classes
    summpg = 0
    vclasscount = 0
    for d in mpg: # iterate over all dictionaries
```

```

        if d['class'] == t: # if the cylinder amount type matches,
            summpg += float(d['hwy']) # add the hwy mpg
            vclasscount += 1 # increment the count
        HwyMpgByClass.append((t, summpg / vclasscount)) # append the tuple
    → ('class', 'avg mpg')

HwyMpgByClass.sort(key=lambda x: x[1])
HwyMpgByClass

```

## 5 The Python Programming Language: Dates and Times

```

[ ]: import datetime as dt
import time as tm

```

time returns the current time in seconds since the Epoch. (January 1st, 1970)

```

[ ]: tm.time()

```

Convert the timestamp to datetime.

```

[ ]: dtnow = dt.datetime.fromtimestamp(tm.time())
dtnow

```

Handy datetime attributes:

```

[ ]: dtnow.year, dtnow.month, dtnow.day, dtnow.hour, dtnow.minute, dtnow.second #
    → get year, month, day, etc. from a datetime

```

timedelta is a duration expressing the difference between two dates.

```

[ ]: delta = dt.timedelta(days = 100) # create a timedelta of 100 days
delta

```

date.today returns the current local date.

```

[ ]: today = dt.date.today()

```

```

[ ]: today - delta # the date 100 days ago

```

```

[ ]: today > today-delta # compare dates

```

## 6 The Python Programming Language: Objects and map()

An example of a class in python:

```

[ ]: class Person:
    department = 'School of Information' #a class variable

    def set_name(self, new_name): #a method
        self.name = new_name
    def set_location(self, new_location):
        self.location = new_location

```



```
[ ]: person = Person()
person.set_name('Christopher Brooks')
person.set_location('Ann Arbor, MI, USA')
print('{} live in {} and works in the department {}'.format(person.name, person.
    ↳location, person.department))
```

Here's an example of mapping the min function between two lists.

```
[ ]: store1 = [10.00, 11.00, 12.34, 2.34]
store2 = [9.00, 11.10, 12.34, 2.01]
cheapest = map(min, store1, store2)
cheapest
```

Now let's iterate through the map object to see the values.

```
[ ]: for item in cheapest:
    print(item)
```

## 7 The Python Programming Language: Lambda and List Comprehensions

Here's an example of lambda that takes in three parameters and adds the first two.

```
[ ]: my_function = lambda a, b, c : a + b
```

```
[ ]: my_function(1, 2, 3)
```

Let's iterate from 0 to 999 and return the even numbers.

```
[ ]: my_list = []
for number in range(0, 1000):
    if number % 2 == 0:
        my_list.append(number)
my_list
```

Now the same thing but with list comprehension.

```
[ ]: my_list = [number for number in range(0,1000) if number % 2 == 0]
my_list
```