Homework 15

Problem 1 (2 points)

- Paste or type the following code into a **separate file called units.py**
- The file should be in the same folder as this notebook.
- Import or load the module and use it to convert a temperature of 212 degrees F to into degrees C.

```
"""
Simple unit conversion module.
Some basic conversions for length and temperature.
"""

ft_to_m = 0.3048
m_to_ft = 3.28084

ft_to_in = 12.0
in_to_ft = 1.0/12.0
mi_to_ft = 5280.0
ft_to_mi = 1.0/5280.0
in_to_mi = 1.0/12.0/5280.0
mi_to_in = 5280.0*12.0

m_to_km  = 0.001
m_to_dm  = 10.0
m_to_cm  = 100.0
m_to_mm  = 1000.0
m_to_um  = 1.0E6  # m to micrometers

def K_to_C(T_K) :
    return T_K - 278.15

def C_to_K(T_C) :
    return T_C + 278.15

def F_to_C(T_F) :
    return (T_F - 32.0)/1.8

def C_to_F(T_C) :
    return T_C*1.8 + 32.0
```

In [1]:
```
import numpy as np
import units as u

temp = 212                  # F
print("{TF} degrees Fahrenheit is equivalent to {TC:.0f} degrees Celcius".format(
    TF = temp,              # F
    TC = u.F_to_C(temp)     # C
))
```

```
212 degrees Fahrenheit is equivalent to 100 degrees Celcius
```

Problem 2 (3 points)

- Write a cylinder class similar to the cube class that we discussed in class.
- Your class should include an `__init__` function and assume default radius and height of one, but allow the user to change these when an object is created from the class.
- The `__init__` function should also print out the dimensions of the cylinder
- Your class should have functions to compute the volume and the surface area (including the ends). Each function should print the results, not just return them.
- Create two cylinder objects: the first with the default radius and height and a second with radius 2 and height 10. Call their respective functions to print the volume and surface area of each object.
- Your outputs should be well-formatted and easy to understand. Your code should be commented.

In [2]:
```

class cylinder():

    def __init__(self, radius = 1, height = 1, units = "meters"):
        # Initialize and store the relative values.
        self.H = height
        self.R = radius
        self.U = units
        print("New cylinder of radius {R} and height {H}".format(R = radius, H = height))

    def cross_section(self):
        # Calculate the cross-section of the cylinder
        return np.pi*self.R**2

    def volume(self):
        # Calculate the volume of the cylinder
        return self.cross_section() * self.H

    def print_volume(self):
        # Print the calculated valume of the cylinder
        print("The volume of this cylinder is {V:.2f} {U} cubed.".format(
            V = self.volume(),
            U = self.U
        ))

    def surface(self):
        # Calculate the surface area of the cylinder
        return 2 * (self.cross_section() + self.H * self.R * np.pi)

    def print_surface(self):
        # Print the calculated surface area of the cylinder
        print("The surface area of this cylinder is {S:.2f} {U} squared.".format(
            S = self.surface(),
            U = self.U
        ))
```

In [3]:

```
cylinders = (cylinder(), cylinder(2, 10))

for cyl in cylinders:
    cyl.print_volume()

for cyl in cylinders:
    cyl.print_surface()
```

```
New cylinder of radius 1 and height 1
New cylinder of radius 2 and height 10
The volume of this cylinder is 3.14 meters cubed.
The volume of this cylinder is 125.66 meters cubed.
The surface area of this cylinder is 12.57 meters squared.
The surface area of this cylinder is 150.80 meters squared.
```

Problem 3 (3 points)

Think of a new problem (could deal with mathematics, physics, biology, engineering, psychology, gospel living, dating, etc.) for which a Python class could be useful.

A class enables multiple instances or cases or members of that class (i.e. objects) to be created as needed. Each object should have a way to:

1. initialize the object with the bare minimum amount of information (using the `__init__` function)
2. store or modify additional information associated with the object
3. retrieve or report information about the object
4. perform some function or task associated with the data about the object

Write a well-commented class that fulfills these requirements. Create at least two objects in that class and demonstrate that each of the above bulleted features works.

In [4]:

```
class Location():
    def __init__(self, building, room_num):
        self.building = building
        self.room_num = room_num

    def get():
        return "Room {R} of the {B}".format(R = self.room_num, B = self.building)
```

In [5]:
```
class Section():

    def __init__(self, time, location):
        self.time = time
        self.location = location
```

In [6]:
```
class Course():

    def __init__(self, id_code, name, professor, credits):
        self.id_code = id_code
        self.name = name
        self.professor = professor
        self.credit = credits

    def get(self):
        return "{ID}: {N}, a {C} credit class taught by {P}".format(
            ID = self.id_code,
            N = self.name,
            C = self.credit,
            P = self.professor
        )
```

In [7]:
```
class Catalog():

    def __init__ (self, course_map):
        if not isinstance(course_map, dict):
            raise TypeError("Schedule must be given as a dict, but was given as a {T}".format(T=type(course_map)))
        self.courses = course_map.keys()
        self.sections = course_map.values()

    def list_courses(self):
        print("The courses offered this upcomming semester are:")
        for c in self.courses:
            print(c.get())

    def list_sections(self, id_code):
        for c in self.courses:
            if self.courses[i].id_code == id_code:
                print(c.get)
                for j in range(len(self.courses[i])):
                    print("Section {I}: {T} in {L}".format(
                        I=j+1,
                        T=self.courses[i][j].time,
                        L=self.courses[i][j].location.get
                    ))
                    return
        print("Course not found")
        return
```

In [8]:

```
#class Class():

    #def __init__ (self, course, section):
```

In [9]:

```
cs142 = Course('CS 142', "Intro to Programming", "Dr. Patrick van Duyse", 3)
cs235 = Course('CS 235', "Data Structures", "Dr. Patrick van Duyse", 2)
TMCB1141 = Location("TMCB", 1141)
TMCB1001 = Location("TMCB", 1001)
cs_catalog = Catalog({
    cs142:[
        Section("MWF 10:00 AM - 10:50 AM", TMCB1141),
        Section("TTh 10:00 AM - 11:15 AM", TMCB1141)
    ],
    cs235:[
        Section("MW 11:00 AM - 11:50 AM", TMCB1141),
        Section("TTh 8:00 AM - 8:50 AM", TMCB1001)
    ]
})
```

In [10]:

```
cs_catalog.list_courses()
#cs_catalog.list_sections("CS 142")
diction = {
    cs142:[
        Section("MWF 10:00 AM - 10:50 AM", TMCB1141),
        Section("TTh 10:00 AM - 11:15 AM", TMCB1141)
    ],
    cs235:[
        Section("MW 11:00 AM - 11:50 AM", TMCB1141),
        Section("TTh 8:00 AM - 8:50 AM", TMCB1001)
    ]
}
diction.keys()
```

```
The courses offered this upcomming semester are:
```

```
CS 142: Intro to Programming, a 3 credit class taught by Dr. Patrick van Duyse
CS 235: Data Structures, a 2 credit class taught by Dr. Patrick van Duyse
```

Out[10]:

```
dict_keys([<__main__.Course object at 0x7f9e9408dbe0>, <__main__.Course object at 0x7f9e9408d6a0>])
```

Note:

I essentially started building a data framework for a university course catalogue.

This ended up being significantly larger than I intended it to be. Is it complete? No, but it's significantly more than was requested by the program.

In [ ]: