```
--------------------------------------------Lab 1
----------------------------------------
1. The Python Interface
# Example, do not modify!
print(5 / 8)

# Put code below here

print(7 + 10)

2.When to use Python?
All of above

3. Adding comments
# Just testing division
print(5 / 8)

# Addition works too
print(7 + 10)

4. Python as a calculator

# Addition and subtraction
print(5 + 5)
print(5 - 5)

# Multiplication and division
print(3 * 5)
print(10 / 2)

# Exponentiation
print(4 ** 2)

# Modulo
print(18 % 7)

# How much is your $100 worth after 7 years?
print(100*1.1**7)
--------------------------------------------Lab 2
----------------------------------------
1. Variable Assignment
# Create a variable savings
savings=100

# Print out savings
print(savings)

2.Calculations with variables
```

```
# Create a variable savings
savings = 100

# Create a variable factor
factor=1.10

# Calculate result
result = savings *factor**7

# Print out result
print(result)
```

3.Other variable types

```
# Create a variable desc
desc ="compound interest"

# Create a variable profitable
profitable=True
```

4.Guess the type

a is of type float, b is of type str, c is of type bool

5. Operations with other types

```
# Several variables to experiment with
savings = 100
factor = 1.1
desc = "compound interest"

# Assign product of factor and savings to year1
year1 = savings*factor

# Print the type of year1
print(type(year1))

# Assign sum of desc and desc to doubledesc
doubledesc= desc+desc

# Print out doubledesc
print(doubledesc)
```

6.Type conversion

```
# Definition of savings and result
savings = 100
result = 100 * 1.10 ** 7

# Fix the printout
print("I started with $" + str(savings) + " and now have $" + str(result) + ".
Awesome!")
```

```
# Definition of pi_string
pi_string = "3.1415926"

# Convert pi_string into float: pi_float
pi_float= float(pi_string)
```

7. Can Python handle everything?
c
---------------------------------------------Lab 3
-----------------------------------------

1. Create a list

```
# area variables (in square meters)
hall = 11.25
kit = 18.0
liv = 20.0
bed = 10.75
bath = 9.50

# Create list areas

areas= [hall,kit,liv,bed,bath]
# Print areas

print(areas)
```

2. Create list with different types

```
# area variables (in square meters)
hall = 11.25
kit = 18.0
liv = 20.0
bed = 10.75
bath = 9.50

# Adapt list areas
areas = ["hallway", hall, "kitchen", kit, "living room", liv, "bedroom", bed,
"bathroom", bath]

# Print areas
print(areas)
```

3. Select the valid list
A

4.List of lists

```
# area variables (in square meters)
```

```
hall = 11.25
kit = 18.0
liv = 20.0
bed = 10.75
bath = 9.50

# house information as list of lists
house = [["hallway", hall],
         ["kitchen", kit],
         ["living room", liv],
         ["bedroom", bed],
         ["bathroom", bath]]

# Print out house
print(house)

# Print out the type of house
print(type(house))
```

---------------------------------------------Lab 4
-----------------------------------------

1. Subset and conquer

```
# Create the areas list
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75,
"bathroom", 9.50]

# Create the areas list
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75,
"bathroom", 9.50]

# Print out second element from areas
print(areas[1])
# Print out last element from areas
print(areas[-1])

# Print out the area of the living room
print(areas[5])
```

2. Subset and calculate

```
# Create the areas list
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75,
"bathroom", 9.50]

# Sum of kitchen and bedroom area: eat_sleep_area
eat_sleep_area = areas[3] + areas[-3]
```

```
# Print the variable eat_sleep_area
print(eat_sleep_area)
```

3. Slicing and dicing

```
# Create the areas list
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75,
"bathroom", 9.50]

# Use slicing to create downstairs
downstairs = areas[0:6]

# Use slicing to create upstairs
upstairs = areas[6:10]

# Print out downstairs and upstairs
print(downstairs)
print(upstairs)
```

4.Slicing and dicing(2)

```
# Create the areas list
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75,
"bathroom", 9.50]

# Alternative slicing to create downstairs
downstairs = areas[:6]

# Alternative slicing to create upstairs
upstairs = areas[6:]
```

5.Subsetting lists of lists

A float: the bathroom area

---------------------------------------------Lab 5
-----------------------------------------

1. Replace list elements

```
# Create the areas list
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75,
"bathroom", 9.50]

# Correct the bathroom area
areas[-1] = 10.50

# Change "living room" to "chill zone"
areas[4] = "chill zone"
```

2.Extend a list

```
# Create the areas list (updated version)
areas = ["hallway", 11.25, "kitchen", 18.0, "chill zone", 20.0,
         "bedroom", 10.75, "bathroom", 10.50]

# Add poolhouse data to areas, new list is areas_1
areas_1 = areas + ["poolhouse", 24.5]

# Add garage data to areas_1, new list is areas_2
areas_2 = areas_1 + ["garage", 15.45]
```

3.Delete list elements

c

4. Inner workings of lists

```
# Create list areas
areas = [11.25, 18.0, 20.0, 10.75, 9.50]

# Create areas_copy
areas_copy = list(areas)

# Change areas_copy
areas_copy[0] = 5.0

# Print areas
print(areas)
```

---------------------------------------------Lab 6
-----------------------------------------

1. Familiar functions

```
# Create variables var1 and var2
var1 = [1, 2, 3, 4]
var2 = True

# Print out type of var1
print(type(var1))

# Print out length of var1
print(len(var1))

# Convert var2 to an integer: out2
out2 = int(var2)
```

2.Help!

complex() takes two arguments: real and imag. real is a required argument, imag is an optional argument.

3. Multiple arguments

```
# Create lists first and second
first = [11.25, 18.0, 20.0]
second = [10.75, 9.50]

# Paste together first and second: full
full = first + second

# Sort full in descending order: full_sorted
full_sorted = sorted(full, reverse = True)

# Print out full_sorted
print(full_sorted)
```

---------------------------------------------Lab 7
-----------------------------------------

1.String Methods

```
# string to experiment with: room
room = "poolhouse"

# Use upper() on room: room_up
room_up = room.upper()

# Print out room and room_up
print(room)
print(room_up)

# Print out the number of o's in room
print(room.count("o"))
```

2.List Methods

```
# Create list areas
areas = [11.25, 18.0, 20.0, 10.75, 9.50]

# Print out the index of the element 20.0
print(areas.index(20.0))

# Print out how often 14.5 appears in areas
print(areas.count(14.5))
```

3. List Methods (2)

```
# Create list areas
areas = [11.25, 18.0, 20.0, 10.75, 9.50]

# Use append twice to add poolhouse and garage size
areas.append(24.5)
```

```
areas.append(15.45)

# Print out areas
print(areas)

# Reverse the orders of the elements in areas
areas.reverse()

# Print out areas
print(areas)
--------------------------------------------Lab 8
-----------------------------------------
1. import packages
# Definition of radius
r = 0.43

# Import the math package
import math

# Calculate C
C = 2 * r * math.pi

# Calculate A
A = math.pi * r ** 2

# Build printout
print("Circumference: " + str(C))
print("Area: " + str(A))

2.Selective import
# Definition of radius
r = 192500

# Import radians function of math package
from math import radians

# Travel distance of Moon over 12 degrees. Store in dist.
dist = r * radians(12)

# Print out dist
print(dist)

3.Different ways of importing

from scipy.linalg import inv as my_inv

--------------------------------------------Lab 9
-----------------------------------------
```

1. Your First Numpy Array

```
# Create list baseball
baseball = [180, 215, 210, 210, 188, 176, 209, 200]

# Import the numpy package as np
import numpy as np

# Create a Numpy array from baseball: np_baseball
np_baseball=np.array(baseball)
# Print out type of np_baseball
print(type(np_baseball))
```

2.Baseball players' height

```
# height is available as a regular list

# Import numpy
import numpy as np

# Create a Numpy array from height: np_height
np_height = np.array(height)

# Print out np_height
print(np_height)

# Convert np_height to m: np_height_m
np_height_m= np_height *0.0254

# Print np_height_m
print(np_height_m)
```

3.Baseball player's BMI

```
# height and weight are available as a regular lists

# Import numpy
import numpy as np

# Create array from height with correct units: np_height_m
np_height_m = np.array(height) * 0.0254

# Create array from weight with correct units: np_weight_kg
np_weight_kg = np.array(weight) * 0.453592

# Calculate the BMI: bmi
bmi = np_weight_kg/np_height_m**2

# Print out bmi
print(bmi)
```

4.Lightweight baseball players

```
# height and weight are available as a regular lists

# Import numpy
import numpy as np

# Calculate the BMI: bmi
np_height_m = np.array(height) * 0.0254
np_weight_kg = np.array(weight) * 0.453592
bmi = np_weight_kg / np_height_m ** 2

# Create the light array
light = bmi < 21

# Print out light
print(light)

# Print out BMIs of all baseball players whose BMI is below 21
print(bmi[light])
```

5. Numpy Side Effects

```
np.array([4, 3, 0]) + np.array([0, 2, 2])
```

6. Subsetting Numpy Arrays

```
# height and weight are available as a regular lists

# Import numpy
import numpy as np

# Store weight and height lists as numpy arrays
np_weight = np.array(weight)
np_height = np.array(height)

# Print out the weight at index 50
print(np_weight[50])

# Print out sub-array of np_height: index 100 up to and including index 110
print(np_height[100:111])
```

---------------------------------------------Lab 10
-----------------------------------------

1. Your First 2D Numpy Array

```
# Create baseball, a list of lists
baseball = [[180, 78.4],
            [215, 102.7],
            [210, 98.5],
            [188, 75.2]]

# Import numpy
import numpy as np

# Create a 2D numpy array from baseball: np_baseball
np_baseball = np.array(baseball)

# Print out the type of np_baseball
print(type(np_baseball))

# Print out the shape of np_baseball
print(np_baseball.shape)
```

2. Baseball data in 2D form

```
# baseball is available as a regular list of lists

# Import numpy package
import numpy as np

# Create a 2D numpy array from baseball: np_baseball
np_baseball = np.array(baseball)

# Print out the shape of np_baseball
print(np_baseball.shape)
```

3.Subsetting 2D Numpy Arrays

```
# baseball is available as a regular list of lists

# Import numpy package
import numpy as np

# Create np_baseball (2 cols)
np_baseball = np.array(baseball)

# Print out the 50th row of np_baseball
print(np_baseball[49,:])

# Select the entire second column of np_baseball: np_weight
np_weight = np_baseball[:,1]

# Print out height of 124th player
```

```
print(np_baseball[123, 0])

4.2D Arithmetic
# baseball is available as a regular list of lists
# update is available as 2D Numpy array

# Import numpy package
import numpy as np

# Create np_baseball (3 cols)
np_baseball = np.array(baseball)

# Print out addition of np_baseball and update

print(np_baseball+update)
# Create Numpy array: conversion
conversion = np.array([0.0254, 0.453592, 1])

# Print out product of np_baseball and conversion
print(np_baseball * conversion)
---------------------------------------------Lab 11
----------------------------------------------------
1. Average versus median

# np_baseball is available

# Import numpy
import numpy as np

# Create np_height from np_baseball
np_height =  np_baseball[:,0]

# Print out the mean of np_height
print(np.mean(np_height))

# Print out the median of np_height
print(np.median(np_height))

2. Explore the baseball data

# np_baseball is available

# Import numpy
import numpy as np

# Print mean height (first column)
avg = np.mean(np_baseball[:,0])
print("Average: " + str(avg))
```

```
# Print median height. Replace 'None'
med = np.median(np_baseball[:,0])
print("Median: " + str(med))

# Print out the standard deviation on height. Replace 'None'
stddev = np.std(np_baseball[:,0])
print("Standard Deviation: " + str(stddev))

# Print out correlation between first and second column. Replace 'None'
corr = np.corrcoef(np_baseball[:,0], np_baseball[:,1])
print("Correlation: " + str(corr))
```

3. Blend it all together

```
# heights and positions are available as lists

# Import numpy
import numpy as np

# Convert positions and heights to numpy arrays: np_positions, np_heights
np_positions = np.array(positions)
np_heights = np.array(heights)

# Heights of the goalkeepers: gk_heights
gk_heights = np_heights[np_positions == 'GK']

# Heights of the other players: other_heights
other_heights = np_heights[np_positions != 'GK']

# Print out the median height of goalkeepers. Replace 'None'
print("Median height of goalkeepers: " + str(np.median(gk_heights)))

# Print out the median height of other players. Replace 'None'
print("Median height of other players: " + str(np.median(other_heights)))
```

---------------------------------------------Lab 12
----------------------------------------------------

1. Line Plot

```
# Print the last item from year and pop

print(year[-1])
print(pop[-1])

# Import matplotlib.pyplot as plt
import matplotlib.pyplot as plt

# Make a line plot: year on the x-axis, pop on the y-axis
```

```
plt.plot(year,pop)
plt.show()
```

2. Line Plot (2): Interpretation
2062

3. Line plot (3)

```
# Print the last item of gdp_cap and life_exp

print(gdp_cap[-1])
print(life_exp[-1])
# Make a line plot, gdp_cap on the x-axis, life_exp on the y-axis
plt.plot(gdp_cap,life_exp)

# Display the plot
plt.show()
```

4.Scatter Plot (1)

```
# Change the line plot below to a scatter plot
plt.scatter(gdp_cap, life_exp)

# Put the x-axis on a logarithmic scale
plt.xscale('log')

# Show plot
plt.show()
```

5.Scatter plot (2)

```
# Import package
import matplotlib.pyplot as plt

# Build Scatter plot
plt.scatter(pop,life_exp)

# Show plot
plt.show()
```

---------------------------------------------Lab 13
----------------------------------------------------

1. Build a histogram (1)

```
# Create histogram of life_exp data
plt.hist(life_exp)

# Display histogram
plt.show()
```

2. Build a histogram (2): bins

```
# Build histogram with 5 bins

plt.hist(life_exp,bins=5)
# Show and clean up plot
plt.show()
plt.clf()

# Build histogram with 20 bins
plt.hist(life_exp,bins=20)

# Show and clean up again
plt.show()
plt.clf()
```

3.Build a histogram (3): compare

```
# Histogram of life_exp, 15 bins
plt.hist(life_exp,bins=15)

# Show and clear plot
plt.show()
plt.clf()

# Histogram of life_exp1950, 15 bins
plt.hist(life_exp1950,bins=15)

# Show and clear plot again
plt.show()
plt.clf()
```

4.Choose the right plot (1)
Histogram

5.Choose the right plot (2)
scatter plot
---------------------------------------------Lab 14
----------------------------------------------------

1. Labels

```
# Basic scatter plot, log scale
plt.scatter(gdp_cap, life_exp)
plt.xscale('log')

# Strings
```

```
xlab = 'GDP per Capita [in USD]'
ylab = 'Life Expectancy [in years]'
title = 'World Development in 2007'

# Add axis labels
plt.xlabel(xlab)
plt.ylabel(ylab)

# Add title
plt.title(title)

# After customizing, display the plot
plt.show()
```

2. Ticks

```
# Scatter plot
plt.scatter(gdp_cap, life_exp)

# Previous customizations
plt.xscale('log')
plt.xlabel('GDP per Capita [in USD]')
plt.ylabel('Life Expectancy [in years]')
plt.title('World Development in 2007')

# Definition of tick_val and tick_lab
tick_val = [1000,10000,100000]
tick_lab = ['1k','10k','100k']

# Adapt the ticks on the x-axis
plt.xticks(tick_val, tick_lab)

# After customizing, display the plot
plt.show()
```

3. Sizes

```
# Import numpy as np
import numpy as np

# Store pop as a numpy array: np_pop
np_pop = np.array(pop)

# Double np_pop
np_pop = np_pop * 2

# Update: set s argument to np_pop
plt.scatter(gdp_cap, life_exp, s = np_pop)
```

```
# Previous customizations
plt.xscale('log')
plt.xlabel('GDP per Capita [in USD]')
plt.ylabel('Life Expectancy [in years]')
plt.title('World Development in 2007')
plt.xticks([1000, 10000, 100000],['1k', '10k', '100k'])

# Display the plot
plt.show()
```

4. Colors

```
# Specify c and alpha inside plt.scatter()
plt.scatter(x = gdp_cap, y = life_exp, s = np.array(pop) * 2, c = col, alpha = 0.8)

# Previous customizations
plt.xscale('log')
plt.xlabel('GDP per Capita [in USD]')
plt.ylabel('Life Expectancy [in years]')
plt.title('World Development in 2007')
plt.xticks([1000,10000,100000], ['1k','10k','100k'])

# Show the plot
plt.show()
```

5. Additional Customizations

```
# Scatter plot
plt.scatter(x = gdp_cap, y = life_exp, s = np.array(pop) * 2, c = col, alpha = 0.8)

# Previous customizations
plt.xscale('log')
plt.xlabel('GDP per Capita [in USD]')
plt.ylabel('Life Expectancy [in years]')
plt.title('World Development in 2007')
plt.xticks([1000,10000,100000], ['1k','10k','100k'])

# Additional customizations
plt.text(1550, 71, 'India')
plt.text(5700, 80, 'China')

# Add grid() call
plt.grid(True)

# Show the plot
plt.show()
```

6.Interpretation

A

---------------------------------------------Lab 15
----------------------------------------------------

1. Equality

```
# Comparison of booleans
print(True==False)

# Comparison of integers
print(-5*15!=75)
# Comparison of strings
print("pyscript" == "PyScript")

# Compare a boolean with an integer
print(True ==1)
```

False,True,False,True

2.Greater and less than

```
# Comparison of integers
x = -3 * 6
print(x>=-10)

# Comparison of strings
y = "test"
print("test"<=y)
# Comparison of booleans
print(True>False)
```

False,True,True

3. and or not (1)

```
# Define variables
my_kitchen = 18.0
your_kitchen = 14.0

# my_kitchen bigger than 10 and smaller than 18?
print(my_kitchen>10 and my_kitchen<18)
# my_kitchen smaller than 14 or bigger than 17?
print(my_kitchen>17 or my_kitchen<14)

# Double my_kitchen smaller than triple your_kitchen?
print(2*my_kitchen<3*your_kitchen)
```

False,True,True

4. and, or, not (2)

False

5. Warmup

medium

6. if

```
# Define variables
room = "kit"
area = 14.0

# if statement for room
if room == "kit" :
    print("looking around in the kitchen.")

# if statement for area
if area >15 :
    print("big place!")
```

7. add else

```
# Define variables
room = "kit"
area = 14.0

# if-else construct for room
if room == "kit" :
    print("looking around in the kitchen.")
else :
    print("looking around elsewhere.")

# if-else construct for area
if area > 15 :
    print("big place!")
else :
    print("pretty small.")
```

8.Customize further: elif

```
# Define variables
room = "bed"
area = 14.0
```

```
# if-elif-else construct for room
if room == "kit" :
    print("looking around in the kitchen.")
elif room == "bed":
    print("looking around in the bedroom.")
else :
    print("looking around elsewhere.")

# if-elif-else construct for area
if area > 15 :
    print("big place!")

elif area >10 :
    print( "medium size, nice!" )
else :
    print("pretty small.")
```

---------------------------------------------Lab 16
----------------------------------------------------

1.CSV to DataFrame (1)

```
# Import pandas as pd
import pandas as pd

# Import the cars.csv data: cars
cars = pd.read_csv("cars.csv")

# Print out cars
print(cars)
```

2.CSV to DataFrame (2)

```
# Import pandas as pd
import pandas as pd

# Fix import by including index_col
cars = pd.read_csv('cars.csv',index_col=0)

# Print out cars
print(cars,)
```

3.Square Brackets

```
# Import cars data
import pandas as pd
cars = pd.read_csv('cars.csv', index_col = 0)

# Print out country column as Pandas Series
```

```
Series =print(cars['country'])
# Print out country column as Pandas DataFrame

DataFrame =print(cars[['country']])
```

4. loc (1)

```
# Import cars data
import pandas as pd
cars = pd.read_csv('cars.csv', index_col = 0)
print(cars)
# Print out observation for Japan
print(cars.loc['JAP'])
# Print out observations for Australia and Egypt
print(cars.loc[['AUS','EG']])
```

5. loc(2)

```
# Import cars data
import pandas as pd
cars = pd.read_csv('cars.csv', index_col = 0)
print(cars)
# Print out drives_right value of Morocco

print(cars.loc['MOR', 'drives_right'])
# Print sub-DataFrame
print(cars.loc[['RU', 'MOR'], ['country', 'drives_right']])
```