Code-Python

#code

CFA PSM

Print:

```
a = "Procter and Gamble (P&G)"

b = "US GAAP"

c = "$500000"

print("{}, which reports using {}, sold a piece of land for {}".format(a,b,c))
```

Input:

```
name = input("Please enter the name of the stock:")

beta = float(input("Enter the stock beta:"))

rf = float(input("Enter the risk-free rate of return in %: "))

r_mkt = float(input("Enter the expected broad market rate of return in %: "))

r_exp = float()

r_exp = (rf + (beta*(r_mkt - rf)))

r_exp
```

List:

```
dividend_companies = ['Pioneer Natural Resources Co. (ticker: PXD)',
                      'Lumen Technologies Inc. (LUMN)',
                      'Altria Group Inc. (MO)',
                      'Vornado Realty Trust (VNO)',
                      'Devon Energy Corp. (DVN)',
                      'AT&T Inc. (T)',
                      'Simon Property Group Inc. (SPG)',
                      'Verizon Communications Inc. (VZ)',
                      'Kinder Morgan Inc. (KMI)']
```

```
# The last element can be accessed using index -1
# Alternatively, you can use len(list)-1
# Alternatively, you can manually count the number of elements and specify the index (note that you need to start counting from index 0)
print(dividend_companies[-1])
print(dividend_companies[len(dividend_companies)-1])
print(dividend_companies[8])
# Print the first 3 elements in the list
print(dividend_companies[0:3])
```

Dictionary:

```
investor_assets = {"Equity": 5000, "Fixed Income": 1000, "Derivatives": 300}
print(investor_assets)
# You access specific dictionary values using the keys
investor_assets["Fixed Income"]
# Let's assume that the investor would like to add alternative investments to their portfolio
# Let's add a new item to an existing dictionary

investor_assets["Alternative Investments"] = 4000
print(investor_assets)

# Remove an item from the dictionary
del investor_assets["Fixed Income"]
print(investor_assets)



# Method 2: TO calculate average of the dictonary
sum(technology_stocks.values())/len(technology_stocks)
```

Strings:

```
welcome_message = "Hello & Welcome to Python Programming Fundamentals Course!"
# The .upper() method is used to convert the string into uppercase
welcome_message.upper()
# The .split() method is used to divide up the string into words
# The output from the .split() method is a list which is denoted by square bracket - []
words = welcome_message.split()
words
# You can specify which letter could be used to perform the split
words = welcome_message.split('&')
words

# Combining two strings together (Let's combine the first and last name of an investor)
first_name = "Sarah"
last_name = "David"
full_name = first_name + last_name
full_name
full_name = first_name + " " + last_name



# The code will get the user input and split it into words separated by space " "
feedback = input("Welcome to the bank, What did we do at the bank today?")
words = feedback.split(" ")
print("Here are the list of words you entered:", words)

# The code will ask the user to enter their e-mail address
# Note that you will need to do two splits, one on the @ and the second split on the '.'
email = input("Please enter your e-mail address:")
output = email.split('@')
```

```
output

# Now we are only interested in the first element in the list with index 0
output[0]

# We will perform the split on the first list element only
name = output[0].split('.')
print(name[0])
```

Comparison Operators:

Conditional Statements(IF-Else):

```
revenue_A = 2000
revenue_B = 2000

if revenue_A > revenue_B:
    print('Company A generates more revenue compared to Company B')
elif revenue_A < revenue_B:
    print('Company B generates more revenue compared to Company B')
else:
    print('Company A generates equal revenue to Company B')



username = input('Welcome to the bank, please enter your username:')
password = input('please enter your password:')

# Note that "and" is a logical operator that generates "True" of both conditions are "True"
if username == 'RyanAhmed' and password =='123$abc':
    print("Access granted")
```

```
else:
    print("Access denied, please try again")
```

Dividend Discount Model:

```
p = float(input("Price: "))
r = float(input("Cost of equity: "))
d = float(input("Enter this year's dividend: "))
g = float(input("Div Growth: "))

d1 = d * (1 + g)
v = d1 / (r - g)

if p > v:
    print("Stock is overvalued, you should not invest in this stock")

elif p < v:
    print("Stock is undervalued, you should invest in this stock")

else:
    print('Stock is fairly valued')
```

FoR Loop:

```
company_names = ['Company A', 'Company B', 'Company C', 'Company D']
for i in company_names:
    print(i)


company_revenues = [600000, 900000, 1000000, 1100000]

total_revenue = 0

for i in company_revenues:
    total_revenue = total_revenue + i

total_revenue


message = 'Welcome to Python Programming Fundamentals Course'
for character in message:
    print(character)



import math
math.prod(my_list) # To multiply each element of a list.
```

Range Function:

```
# range() generates a list of numbers that are used to iterate over "For" loops
# The last integer generated by range() is up to, but not including, the last element
# Example: range(0, 4) generates integers from 0 up to, but not including, 4
for i in range(0, 4):
    print(i)

# Note that range indexing starts at 0 by default
# range() is 0-indexed based meaning that numbers start at 0 and not 1.
for i in range(4):
    print(i)
# Note that you can also add an optional parameter to indicate the step size
for i in range(0, 4, 2):
    print(i)



# Let's define two lists containing the company names along with their corresponding revenues for a
given year
company_names    = ['Company A', 'Company B', 'Company C','Company D']
company_revenues = [600000, 900000, 1000000, 1100000]


# Let's use range() to print out the company names and their corresponding revenues
for i in range(len(company_names)):
    print('index = {}'.format(i))
    print('The Revenue of {} is = ${}'.format(company_names[i], company_revenues[i]))


# Let's assume we want to print companies that only have even index within the list
for i in range(0, len(company_names), 2):
    print('index = {}'.format(i))
    print('The Revenue of {} is = ${}'.format(company_names[i], company_revenues[i]))



# Let's assume we want to print out the list in a reversed order
# We can use the "reversed" function along with range
# Note that we started with index 4 until index 0
for i in reversed(range(len(stocks_names))):
    print('index = {}'.format(i))
    print('The price of {} stock = ${}'.format(stocks_names[i], stock_prices[i]))
    # print('\n')



for i in reversed(range(0, len(stocks_names), 2)):
    print('index = {}'.format(i))
    print('The price of {} stock = ${}'.format(stocks_names[i], stock_prices[i]))
```

While Loop:

```
# Now let's assume that we put the same $1000 in an account that pays 10% annually and leave it for 3 years
# We would like to see what would be the future value after each year
# We can use while loops to iterate over a pre-defined number of years
# You can confirm your answer using the future value calculator: https://www.calculator.net/future-value-calculator.html

years         = 1 # initial value
amount        = 1000
interest_rate = 0.1

while years <= 3:
    amount = amount + (interest_rate * amount)
    print('The Future value in year {} = ${}'.format(years, amount))
    years = years + 1

# "While True" could be used to repeat a block of code forever!
# "break" is used to break the loop
# Let's assume we want to keep doubling a variable with every iteration until the total value reaches $1000 and then we exit
# Define the starting dollar amount = 1

x = 1

while True:
    x = x * 2 # Double the amount
    print('Value = {}'.format(x))
    if x > 1000:
        break



# Note that amount is the Present Value
# You can name the variable "present_value" instead of "amount"

amount = float(input("Enter the amount of funds you would like to invest today in dollars (Present Value (PV)): "))
interest = float(input("Enter the annual interest rate: "))
future_value = float(input("Enter the total dollar amount you would like to have in the future (Future Value (FV)): "))

years = 1

print('The Present Value (PV) in year 0 (Now) = ${}'.format(round(amount, 1)))


while True:
    amount = amount + (interest * amount)
    print('Future Value (FV) in year {} = ${}'.format(years, round(amount, 1)))
    # Condition to break the loop here
    if amount > future_value:
        print('It would take around {} years to reach a future value greater than {}'.format(years,
future_value))
        break
```

Functions:

```
# Let's define a function named "calculate_FV" that takes in:
    # 1. present value
    # 2. interest rate
    # 3. number of years
    # 4. number of compounding periods per year

# The function returns the future value
def calculate_FV(PV, i, n, m):
    return PV * ( 1 + (i / m)) ** (m * n)



# Let's calculate the future value of money given:
    # present value = $100,000
    # interest rate (i) = 10%
    # number of years (n) = 5
    # interest is compounded annually (ie: m = 1)

# Let's confirm our answer using this online calculator:
# https://www.calculatorsoup.com/calculators/financial/future-value-investment-calculator.php
# Test the function using a different interest rate of 15%
```

```
present_value = float(input("Enter the present value: "))

interest = float(input("Enter the annual interest rate: "))

num_years = float(input("Enter the number of years: "))

compounding_periods = float(input("Enter the number of compounding periods: "))

# This is the function call

FV = calculate_FV(present_value, interest, num_years, compounding_periods)

print('The FV of ${} after {} years and {} compounding periods per year at {}% interest rate = $
{}'.format(present_value,
                                                                    num_years,
                                                                    compounding_periods,
                                                                    interest * 100,
                                                                    round(FV, 2)))

# Let's assume that the default number of years is 5 (fixed term interest)

# Upon function definition, we can set a default value for one or more of the function arguments

# If the default value is set, we don't have to pass the year argument to the function every time we call
it.

# if year is not provided, the default value will be used

# if year is povided, it would override the existing default value


def calculate_FV(PV, i, m, n = 5):

    return PV * ( 1 + (i / m)) ** (m * n)
```

Built-in Functions:

```
# Note that spaces are for code readabilty and they won't make a difference in the output

A = [-10, -30, -80.6, 70, 21.9]

B = [-3, 7.3, 4.7, 6, 8]
```

```
C = abs( sum(A + B) )
C
```

Pandas:

```
import pandas as pd
# Let's define a two-dimensional Pandas DataFrame from a python dictionary
# We will use Pandas DataFrame constructor method "pd.DataFrame()" to create our Pandas DataFrame
# Data Source: https://statisticstimes.com/economy/projected-world-gdp-ranking.php

GDP_df = pd.DataFrame({'Country ID': ['USA', 'CHN' , 'IND', 'ARE', 'CAN', 'MEX'],
                       'Country':['United States', 'China', 'India', 'United Arab Emirates', 'Canada', 'Mexico'],
                       'GDP Per Capita [$]':[69375, 11891, 2116, 43538, 52791, 9967],
                       'Global Rank':[5, 64, 150, 24, 15, 72]})
GDP_df

# Let's obtain the data type of this pandas DataFrame
type(GDP_df)
# Let's view the first couple of rows using ".head()" method
GDP_df.head(2)
# Let's view the last couple of rows using ".tail()" method
GDP_df.tail(3)
# You can access a specific column in the Pandas DataFrame using the header name
GDP_df['GDP Per Capita [$]']
# Let's obtain a statistical summary about the DataFrame using the "describe()" method
GDP_df.describe()
# Obtain DataFrame information using the info() method
GDP_df.info()
```

Missing Data:

<<Handling Missing Data with Pandas.html>>

DataFrame Filtering & Sorting:

<<DataFrame Filtering and Sorting.html>>

DataFrames with Functions:

<<DataFrames with Functions.html>>

pandas DataFrames Concatenation and Merging:

<<DataFrames Concatenation and Merging.html>>

GET FINANCIAL MARKETS DATA:

<<Get Financial Markets Data.html>>

Capstone Project :

<<Final Capstone Project.ipynb>>

<<Final Capstone Project.html>>

<<Module 1, Unit 5, Lesson 1.zip>>