**Week 3 In-Class Exercises (Conditions and Tuples)**

**Q1: Code Tracing [ \* ]**

Take a look at the following code. What do you think will be the output of the code?

a = 20

b = 20

if a >= b:

print("a >= b")

if a <= b:

print("a <= b")

a = 30

b = 30

if a >= b:

print("a >= b")

elif a <= b:

print("a <= b")

c = "IS111"

d = "is111"

e = "IS" + "111"

print(c == d)

print(c == e)

def test\_if\_else(condition1, condition2):

if (condition1):

print("Great!")

return True

elif(condition2):

print("Good!")

return True

else:

print("Okay")

return False

result = test\_if\_else(4 % 2 != 0, 3 // 2 == 1)

print(result)

**Q2: Day of a Week [ \* ]**

1. Assume the days of a week are numbered 0, 1, 2, 3, 4, 5 and 6 from Sunday to Saturday. Write a function called get\_day\_of\_week which takes in a number between 0 and 6 (both inclusive) and returns a string that represents the corresponding day of a week. For example, if the number passed to the function is 4, the function should **return** 'Thursday'.
2. Prompt the user for a number between 0 and 6 (both inclusive). Assume that the user is always going to enter an integer. Call the function above and then display on the screen the corresponding day of a week.
   1. If the user enters a number below 0, display the following error message: 'Your number should be at least 0.'
   2. If the user enters a number above 6, display the following error message: 'Your number should be at most 6.'

Two sample runs of the code can be found below:





**Q3: Nested if/else [ \* ]**

What values of **a**, **b**, **c** and **d** can cause the code below to display the following output?

* Alpha
* Beta
* Gamma
* Delta

if a and b:

if c:

print('Alpha')

else:

print('Beta')

else:

if d:

print('Gamma')

else:

print('Delta')

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| output | value of a | value of b | value of c | value of d |
| Alpha |  |  |  |  |
| Beta |  |  |  |  |
| Gamma |  |  |  |  |
| Delta |  |  |  |  |

**Q4: Tax Calculator [ \*\* ]**

Recall that last week we had an exercise on tax calculation. This week we will use if/else to solve the problem, which is much easier than using the max() function to solve the problem.

Refer to the table below for the income tax rates:

|  |  |  |
| --- | --- | --- |
| **Chargeable Income** | **Income Tax Rate (%)** | **Gross Tax Payable ($)** |
| First $20,000  Next $10,000 | 0  2 | 0  200 |
| First $30,000  Next $10,000 | -  3.50 | 200  350 |
| First $40,000  Next $40,000 | -  7 | 550  2,800 |
| First $80,000  Next $40,000 | -  11.5 | 3,350  4,600 |
| First $120,000  Next $40,000 | -  15 | 7,950  6,000 |
| First $160,000  Next $40,000 | -  18 | 13,950  7,200 |
| First $200,000  Next $40,000 | -  19 | 21,150  7,600 |
| First $240,000  Next $40,000 | -  19.5 | 28,750  7,800 |
| First $280,000  Next $40,000 | -  20 | 36,550  8,000 |
| First $320,000  In excess of $320,000 | -  22 | 44,550 |

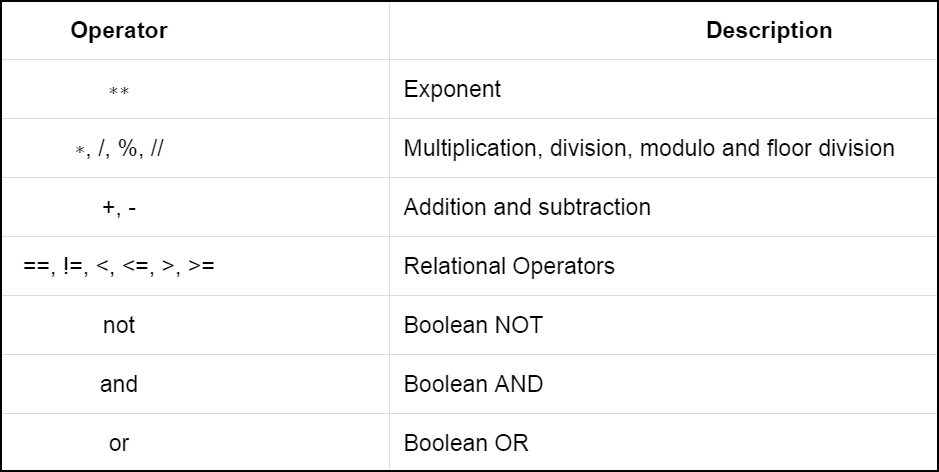
1. Write a function called calculate\_income\_tax() that takes in the annual taxable income of a person and returns the total tax the person has to pay. The function should be able to handle any income value of 0.0 and above. Use conditional statements to help you with the implementation.
2. Prompt the user for his/her annual taxable income and display the tax he/she has to pay. Two sample runs are shown below.

**Q5: Evaluation of Expressions [ \*\* ]**

What are the values of the following expressions?

* not True or (3 >= 3 or 9 < 4) and False
* not True or 3 >= 3 or 9 < 4 and False
* False == (False or not True) or not (2 \* 4 % 3 == 1)

You may want to check the Operator Precedence table provided below to evaluate the expressions.



**Q6: De Morgan’s Law [ \*\* ]**

Simplify the following expression.

not ( not ( a and not b ) or ( b or not a ) )

Simplified expression: .

Now check your answer by plugging in all combinations of possible values of a and b and evaluating the expression above as well as your simplified expression. Fill in your evaluations in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| **a** | **b** | **not (not (a and not b) or (b or not a))** | **Your simplified expression** |
| True | True |  |  |
| True | False |  |  |
| False | True |  |  |
| False | False |  |  |



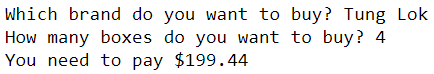
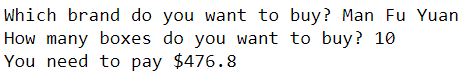
**Q7: Mooncakes [ \*\* ]**

A shop that sells mooncakes offers discount based on how many boxes of mooncakes a customer is going to buy.

* The shop sells the following two brands of mooncakes:
  + Tung Lok: $55.40 / box
  + Man Fu Yuan: $59.60 / box
* If a customer buys 5 or more boxes of mooncakes of the same brand, a discount of 20% will be applied.
* If a customer buys between 2 and 4 boxes of mooncakes of the same brand, a discount of 10% will be applied.
* If a customer buys only 1 box of mooncakes, no discount is applied.

1. Write a function called get\_discount\_rate. This function takes in a parameter called num\_boxes that indicates how many boxes of mooncakes of a particular brand a customer wants to buy. The function returns the discount rate. For example, if num\_boxes is 6, the function returns 0.2. The function returns 0.0 if no discount is offered.
2. Write another function called calculate\_total\_amount. This function takes in two parameters: (1) brand, which is a string indicating the brand of mooncakes, and (2) num\_boxes, which indicates the number of boxes a customer wants to buy. The function returns the total amount the customer has to pay for buying that number of boxes of mooncakes of that brand.
3. Prompt the customer for the brand of mooncakes he/she wants to buy and the number of boxes he/she wants to buy. Display the amount the customer has to pay. You can assume that the customer is always going to enter either “Tung Lok” or “Man Fu Yuan” as the brand and an integer as the number of boxes.

Two runs of the program are shown below:



**Q8: Honey [ \*\* ]**

A shop sells honey in two sizes of jars. A 500g jar is priced at $58.50, while a 1kg jar of honey is priced at $98.50.

1. Inside a file called **retail\_utility.py**, write a function called calculate\_max\_quantity\_and\_change. This function takes in two parameters:
   1. unit\_price: The unit price of an item (e.g., $58.50).
   2. amount: The total amount of money a customer wants to spend to buy that item.

This function **returns** a tuple that contains two values: (1) The maximum quantity of that item the customer can buy with the specified amount of money. (2) The change (remaining amount of money) the customer has after buying the maximum quantity of that item. For example, if unit\_price is $58.50 and amount is $130.00, then the function should return 2 as the maximum quantity and $13.00 as the change, and hence it should return the tuple (2, 13.0).

1. Inside another file called **purchase\_honey.py**, prompt the user for the amount of money he/she wants to spend to buy honey from this shop. Assume that the user wants to maximize the total amount of honey he/she could buy. Use the function you’ve implemented above to calculate the total amount of honey he/she could buy and the remaining amount of money he/she has. Use the function you have implemented in **retail\_utility.py** to help you.

Two sample runs can be found below:





**Q9: Change [\*\*]**

Assume that a store in Singapore has (only) the following bank notes and coins:

Bank notes: $2, $5, $10

Coins: 1 cent, 5 cents, 10 cents, $1

Write a program that computes the number of minimum number of notes and coins, for a given change, to be returned to the customer.

Here is an example of how the output may look like. Notice that the bank note/coin is printed out only if there is at least one of that bank note/coin to be returned as change to the customer.

What is the change? $53.84

$10 note: 5

$2 note: 1

$1 coin: 1

10 cents coin: 8

1 cent coin: 4

You will need to use the modulo (remainder) and floor division functions in your answer.

You should have no more than 40 lines of code in your answer (excluding blank lines).