Course › powerf... › Practice › Mod02...

Mod02 Practice

Module 2 Practice

The Jupyter Notebook Library Source is located at https://notebooks.azure.com/eric/libraries/Dev330x

Students will be able to:

3-2.1 Boolean Expressions and Compound Conditionals

- Describe the fundamental Boolean operators (and, or, not)
- Use Boolean operators to combine comparisons
- Recognize that different Boolean expressions can yield equal results
- Employ combined comparisons to control program flow (i.e. `if` statements)

3-2.2 Advanced Loop Structures

- Recognize the purpose of a `pass` statement
- Differentiate between `break` and `continue` statements
- Control loop iteration using `break` or `continue`
- Use nested loops to iterate over the elements of a table
- Employ compound conditional expressions in a loop structure

3-2.3 Containment, Identity, and Operator Precedence

- Test if a list contains a certain element
- Test if a string is contained in another string
- Test the identity of objects (i.e. int, float, lists, string...etc)

- Recognize the effect of different operator precedence (including: assignment (=), relational (<, >=,...), Boolean (and, or, not), arithmetic (/ // % * + -), identity (is) , and containment (in))

3-2.4 Powerful Output Formatting

- Format strings using old-style `printf` formatting, this includes:
  - Formatting numbers
  - Formatting strings
  - Padding
  - Alignment
- Format strings using new style formatting, this includes:
  - Formatting numbers
  - Formatting strings
  - Padding
  - Alignment

Task 1

Compound Conditionals

User input

```
# [ ] Complete the following program to validate a user input is o
# The range limits (-50, 0, 100, 200) are not valid inputs
# Valid test inputs include: -55, 20, 99
# Invalid test inputs include: -30, 150 (and the range limts -50, 

# User input
x = input("Enter a number outside the ranges [-50, 0] and [100, 20

# convert x to int
x = int(x)

# Test input validity
```

```
## [ ] Repeat the previous task using a different conditional stat
# Complete the following program to validate a user input is outsi
# The range limits (-50, 0, 100, 200) are not valid inputs

# User input
x = input("Enter a number outside the ranges [-50, 0] and [100, 20

# convert x to int
x = int(x)

# Test input validity
```

```
# [ ] Complete the following program to validate a user input is *
# The range limits (-50, 0, 100, 200) are not valid inputs
# [ ] Complete the following program to validate a user input is d
# inside either the range [-75, -25] or the range [50, 75]
# The range limit (75) is a valid input; whereas, (-50, -25, 50) a
# Valid test inputs include: -33, -60, 63
# Invalid test inputs include: -6, 33 (out of range), -29, 55 (not
```

Tax brackets

Calculating taxes is a complicated task; however, for the purpose for this task we will consider a simple model that follows this table:

Rate	Taxable Income	Tax Owed
10%	$0 to $9,325	10% of Taxable Income
15%	$9,325 to $37,950	$932.50 plus 15% of the excess over $9,325
25%	$37,950 to $91,900	$5,226.25 plus 25% of the excess over $37,950
28%	$91,900 to $191,650	$18,713.75 plus 28% of the excess over $91,900
33%	$191,650 to $416,700	$46,643.75 plus 33%% of the excess over $191,650
35%	$416,700 to $418,400	$120,910.25 plus 35% of the excess over $416,700
39.60%	$418,400+	$121,505.25 plus 39.6% of the excess over $418,400

```
# [ ] The following program prompts the user for a taxable income
# Complete the function `tax_owed(income)` using conditional state

# Test cases:
# income = 5000,   Taxes owed = $ 500.0
# income = 10000,  Taxes owed = $ 1033.75
# income = 40000,  Taxes owed = $ 5738.75
# income = 100000, Taxes owed = $ 20981.75
# income = 200000, Taxes owed = $ 49399.25
# income = 417000, Taxes owed = $ 121015.25
# income = 500000, Taxes owed = $ 153818.85

def tax_owed(income):
    """
    Calculate the taxes owed using the tax bracket table.

    args:
        income: Taxable income in dollars

    returns:
        tax_owed: Taxes owed based on income and tax bracket
    """
    #TODO
    pass


# Prompt for taxable income
x = int(input("Enter the taxable income: "))
tax = tax_owed(x)
print("Taxes owed = $", tax)
```

Task 2

Loops: `break`, `continue`

```
# [ ] The following program checks if `lst` is symmetric around it
# In other words, it tests if the first element equals the last, t
#
# `lst` contains 10001 numbers, it was created using a symmetric l
# In the current form of the program, the loop needs to iterate 50
#
# Modify the program to improve its efficiency and reduce the numb

lst = [0, 9247, 30629, 48757, 28498, 31681, 17183, 11678, 10402, 3

iteration_count = 0
symmetric = True
center = len(lst) // 2; # int division (//) is used to avoid fract
for i in range(center):
    if lst[i] != lst[len(lst) - i - 1]:
        symmetric = False
    iteration_count = iteration_count + 1

print("Number of iterations:", iteration_count)

if symmetric:
    print("The list is symmetric")
else:
    print("The list is NOT symmetric")
```

```
# [ ] A palindrome is a sequence of characters which reads the sam
# Complete the function `is_palindrome` to test if the input strin


def is_palindrome(word):
    """
    Test if word is a palindrome.

    Input:
    word: string to be tested

    Returns:
    palindrome: Boolean variable containing True if word is a pali
    """
    pass


# Test cases
w = "madam"
#w = "sir"

if (is_palindrome(w)):
    print(w, "is a palindrome")
else:
    print(w, "is NOT a palindrome")
```

User input

```
# In a previous task, you completed a program to validate a user i
#
# Write a program to print out all of the possible valid numbers
```

```
# [ ] Write a program to prompt the user for an input outside the
# If the user input is invalid, the program prompt the user again
# The range limits (-50, 0, 100, 200) are not valid inputs
```

Task 3

Nested loop

Tables

In this task, you will manipulate the same table you saw before:

5	2	6
4	6	0
9	1	8
7	3	8

```
# [ ] Write a program to display the transpose of `table`.
# In other words, display the rows as columns and the columns as r
# the first row across will be: 5   4   9   7

table = [[5, 2, 6], [4, 6, 0], [9, 1, 8], [7, 3, 8]]
```

```
# [ ] Write a program to display the sum of each column in `table`

table = [[5, 2, 6], [4, 6, 0], [9, 1, 8], [7, 3, 8]]
```

```
# [ ] Write a program to display the numbers in `table` as a singl
# output: 5 2 6 4 6 0 9 1 8 7 3 8

table = [[5, 2, 6], [4, 6, 0], [9, 1, 8], [7, 3, 8]]
```

```
# [ ] Write a program to count the number of odd numbers in `table

table = [[5, 2, 6], [4, 6, 0], [9, 1, 8], [7, 3, 8]]
```

Character art

```
# [ ] Complete the function `generate_diamond` so it displays a di
# *NOTE*: The `size` should be odd, otherwise you should subtract 
# For size = 11, the star should look like:
#     *
#    * *
#   *   *
#  *     *
# *       *
#*         *
# *       *
#  *     *
#   *   *
#    * *
#     *

def generate_diamond(size):
    pass
    #TODO

# Display diamond
generate_diamond(11)
```

Task 4

Containment `in`, `not in`

User input

```
# [ ] Write a program to prompt the user for an input from a prede
# If the user input is invalid, the program prompts the user again

valid_nums = [1, 2, 8, 16, 32, 64]
```

Employee records

```
# [ ] The `records` list contains information about some company's
# each of the elements in `records` is a list containing the name
# Write a program that prompts the user for a name and return the

records = [['Colette', 22347], ['Skye', 35803], ['Alton', 45825],
```

Vowel counter

```
# [ ] Complete the `vowel_counter` function below so it returns th

def vowel_counter(sentence):
    """
    Count the number of vowels (AEIOU) in sentence.

    args:
        sentence: string containing vowels to be counted

    returns:
        Number of vowels in sentence
    """
    #TODO
    pass
```

Task 5

Identity `is`, `is not`

```
# [ ] Complete the function `equal_or_identical` to test and print


def equal_or_identical(x, y):
    pass
    #TODO: test x and y for identity and equality

# Test cases
# equal & identical
i1 = [5, 3]
i2 = i1
print("i1, i2")
equal_or_identical(i1, i2)

print()
# equal but NOT identical
e1 = [1, 2]
e2 = [1, 2]
print("e1, e2")
equal_or_identical(e1, e2)

print()
# NOT equal or identical
v1 = [1, 2]
v2 = [5, 3]
print("v1, v2")
equal_or_identical(v1, v2)

# Output should look like:
#i1, i2
#Variables are identical and equal

#e1, e2
#Variables are equal but NOT identical

#v1, v2
#Variables are neither identical nor equal
```

Task 6

Operator precedence

There are many solutions to each of the following problems

```
# [ ] Correct the following expression so the answer is `True`
x = 5

(10 < x < -10) == True
```

```
# [ ] Correct the following expression so the answer is `False`

x = 5
y = -23

x > y or x + y > 0 == True
```

```
# [ ] Correct the following expression so the answer is `True`

2 ** 4 - 3 % 2 == 0
```

Task 7

Output formatting

Temperature conversion tables

```
# [ ] Use old-style formatting and the function `Fahrenheit2Celsiu
# The Fahrenheit temperature goes from 0 to 130 with an increment

#  0 (F) | -17.78 (C)
#  1 (F) | -17.22 (C)
#  2 (F) | -16.67 (C)
#  3 (F) | -16.11 (C)
#  4 (F) | -15.56 (C)
#  5 (F) | -15.00 (C)
#  6 (F) | -14.44 (C)
#    .        .
#    .        .
#    .        .
#126 (F) | +52.22 (C)
#127 (F) | +52.78 (C)
#128 (F) | +53.33 (C)
#129 (F) | +53.89 (C)
#130 (F) | +54.44 (C)

def Fahrenheit2Celsius(f):
    """Convert f Fahrenheit to its Celsius equivalent"""
    return (f - 32) * 5 / 9

#TODO: print conversion table
```

```
# [ ] Use Python-style, .format(), formatting and the function `Ce
# The Celsius temperature goes from 0 to 100 with an increment of

#  0.0 (C) |  32.00 (F)
#  0.5 (C) |  32.90 (F)
#  1.0 (C) |  33.80 (F)
#  1.5 (C) |  34.70 (F)
#  2.0 (C) |  35.60 (F)
#  2.5 (C) |  36.50 (F)
#  3.0 (C) |  37.40 (F)
#  3.5 (C) |  38.30 (F)
#  4.0 (C) |  39.20 (F)
#   .          .          <- don't print the dots, fill in the en
#   .          .
#   .          .
# 97.5 (C) | 207.50 (F)
# 98.0 (C) | 208.40 (F)
# 98.5 (C) | 209.30 (F)
# 99.0 (C) | 210.20 (F)
# 99.5 (C) | 211.10 (F)
#100.0 (C) | 212.00 (F)

def Celsius2Fahrenheit(c):
    """Convert c Celsius to its Fahrenheit equivalent"""
    return c * 9 / 5 + 32

#TODO: print conversion table
```

Grocery receipt

```
# [ ] The `items` list contains information about a purchase trans
# Each item in the list is another list containing:
#    1) Item description
#    2) Price per item
#    3) Quantity purchased

# Write a program to generate a receipt similar to the one below:
# 1) The first line should contain the current date and time (HINT
# 2) Second line should be a dashed line
# 3) Each of the items will be displayed on 2 lines:
#     I) First line contains: item number, description, total item
#    II) Second line contains: quantity purchased, followed by pri
# 4) Line before last should be a dashed line
# 5) Last line should contain the word TOTAL followed by the recei


#Sun December 12, 2017 @ 04:56 P
#--------------------------------
#
#1 - APPLES 1LB          $ 3.98
#    2.0 @ $ 1.99
#2 - OLIVE OIL           $10.99
#    1.0 @ $10.99
#3 - TOMATOS 1LB         $ 3.35
#    2.6 @ $ 1.29
#4 - MILK 1/2G           $ 3.45
#    1.0 @ $ 3.45
#5 - FLOUR 5LB           $ 2.99
#    1.0 @ $ 2.99
#6 - BELL PEPPERS 1LB    $ 3.78
#    2.8 @ $ 1.35
#7 - WHITE TUNA          $ 1.69
#    1.0 @ $ 1.69
#8 - CHEESE 1/2LB        $ 9.98
#    2.0 @ $ 4.99
#--------------------------------
#                 TOTAL $ 40.21



items = [["APPLES 1LB", 1.99, 2], ["OLIVE OIL", 10.99, 1], ["TOMAT

#TODO
```

Module 2 Project

Tic Tac Toe!

```
# [ ] This project is an implementation a Tic Tac Toe game.
# The logic of the game is in the `main` function, read it before

# Use the description and examples under each of the following fun
# 1) draw(board)
# 2) available(location, board)
# 3) mark(player, location, board)
# 4) check_win(board)
# 5) check_tie(board)

from IPython.display import clear_output #to clear the output (spe
from random import randint

def draw(board):
    """
    Draw the `board` table.

    The board reflects the current state of the game, a number ind

    args:
        board: 3x3 table (list of lists) containing the current st

    returns:
        None

    examples:
        At the beginning of the game: board = [['7', '8', '9'], ['
        The printout should look like:

         7 | 8 | 9
        -----------
         4 | 5 | 6
        -----------
         1 | 2 | 3

         After a few marks: board = [['7', '8', 'X'], ['O', 'O', '
         The printout should look like:
         7 | 8 | X
        -----------
         O | O | 6
        -----------
         1 | X | 3
    """
    #TODO
    pass
```

```
def available(location, board):
    """
    Check the availability of a `location` on the current `board`

    An available location on the board contains a number between 1
    If the location contains 'X' or 'O', the location is not avail
    otherwise, the function should return True indicating the loca

    args:
        location: a number between 1 and 9 stored as a string
        board: 3x3 table (list of lists) containing the current st

    returns:
        True if the location is available. False if the location i

    examples:
        At the beginning of the game: board = [['7', '8', '9'], ['
        The printout should look like:

         7 | 8 | 9
        -----------
         4 | 5 | 6
        -----------
         1 | 2 | 3

         available("1", board) --> returns True
         available("9", board) --> returns True


         After a few marks: board = [['7', '8', 'X'], ['O', 'O', '
         The printout should look like:
         7 | 8 | X
        -----------
         O | O | 6
        -----------
         1 | X | 3

         available("1", board) --> returns True, because there is
         available("5", board) --> returns False, because there is
         available("9", board) --> returns False, because there is
    """
    #TODO
    pass
```

```
def mark(player, location, board):
    """
    Mark `location` on the `board` with the `player` symbol.

    Should replace the `location` number on the board with `X` or

    args:
        player: player's symbol, either 'X' or 'O'
        location: a number between 1 and 9 stored as a string
        board: 3x3 table (list of lists) containing the current st

    returns:
        None

    examples:
        At the beginning of the game: board = [['7', '8', '9'], ['
        The printout should look like:

         7 | 8 | 9
        -----------
         4 | 5 | 6
        -----------
         1 | 2 | 3

         After mark('O', '4', board)
         The printout should look like:
         7 | 8 | 9
        -----------
         O | 5 | 6
        -----------
         1 | 2 | 3

         After mark('X', '3', board)
         The printout should look like:
         7 | 8 | 9
        -----------
         O | 5 | 6
        -----------
         1 | 2 | X

         After mark('O', '9', board)
         The printout should look like:
         7 | 8 | O
        -----------
         O | 5 | 6
        -----------
         1 | 2 | X
    """
    #TODO
    pass

def check_win(board):
    """
    Check if there is a winner.

    A win happens if the either of the players was able to place 3
    a horizontal, vertical, diagonal, or anti-diagonal placement.

    args:
        board: 3x3 table (list of lists) containing the current st

    returns:
        True if there is a winner. False if there is no winner yet

    examples:
        Horizontal win:
        ================

         7 | O | 9
        -----------
         X | X | X
        -----------
         1 | O | 3
        check_win(board) --> returns True, because 'X' won


         O | O | O
        -----------
         X | X | 6
        -----------
         X | O | 3
        check_win(board) --> returns True, because 'O' won


        Vertical win:
        ================

         7 | 8 | X
        -----------
         X | O | X
        -----------
```

```
 O | O | X
check_win(board) --> returns True, because 'X' won


 X | O | O
-----------
 4 | O | 6
-----------
 X | O | X
check_win(board) --> returns True, because 'O' won


Diagonal win:
================

 X | 8 | O
-----------
 4 | X | X
-----------
 O | O | X
check_win(board) --> returns True, because 'X' won


 O | X | O
-----------
 X | O | X
-----------
 1 | 2 | O
check_win(board) --> returns True, because 'O' won


Anti-Diagonal win:
================

 O | 8 | X
-----------
 4 | X | X
-----------
 X | O | O
check_win(board) --> returns True, because 'X' won


 7 | 8 | O
-----------
 X | O | X
-----------
```

```
         O | O | X
        check_win(board) --> returns True, because 'O' won


        No winners yet:
        ================

         O | 8 | 9
        -----------
         4 | X | X
        -----------
         X | O | O
        check_win(board) --> returns False
    """
    #TODO
    pass

def check_tie(board):
    """
    Check the game for a tie, no available locations and no winner

    args:
        board: 3x3 table (list of lists) containing the current st

    returns:
        True if there is a tie. False the board is not full yet or

    examples:

         O | O | X
        -----------
         X | X | O
        -----------
         O | O | X

        check_tie(board) --> returns True



         O | O | 9
        -----------
         X | X | 6
        -----------
         X | O | 3

        check_tie(board) --> returns False, because there are stil
    """
    #TODO
    pass

def dashes():
    """Print a fancy line of dashes"""
    print("o" + 35 *'-' + "o")

def display(message):
    """
    Print `message` in the center of a 35 characters string

    args:
        message: string to display

    returns:
        None
    """
    print("|{:^35s}|".format(message))

def main():
    # initializing game
    board = [['7', '8', '9'], ['4', '5', '6'], ['1', '2', '3']]
    # select the first player randomly
    player = ['X', 'O']
    turn = randint(0, 1)

    win = False
    tie = False
    while(not win and not tie):
        # switch players
        turn = (turn + 1) % 2
        current_player = player[turn] # contains 'X' or 'O'

        clear_output()

        # display header
        dashes()
        display("TIC TAC TOE")
        dashes()

        # display game board
        print()
        draw(board)
        print()
        # display footer
        dashes()
        # player select a location to mark
        while True:
            location = input("|{:s} Turn, select a number (1, 9):
            if available(location, board):
                break # Only the user input loop, main loop does N
            else:
                print("Selection not available!")
        dashes()

        # mark selected location with player symbol ('X' or 'O')
        mark(current_player, location, board)

        # check for win
        win = check_win(board)

        # check for tie
        tie = check_tie(board)


    # Display game over message after a win or a tie
    clear_output()

    # display header
    dashes()
    display("TIC TAC TOE")
    dashes()

    # display game board (Necessary to draw the latest selection)
    print()
    draw(board)
    print()

    # display footer
    dashes()
    display("Game Over!")
    if(tie):
        display("Tie!")
    elif(win):
        display("Winner:")
        display(current_player)
    dashes()


# Run the game
```

```
main()
```