Intro to Computer Science
CS-UH 1001, Spring 2022

Lecture 6 – Logical Operators, Count-controlled Loops, Nested Loops, Multi-dimensional Lists

Today's Lecture

- Logical Operators
- Count-controlled Loops
- Nested Loops
- Multi-Dimensional Lists

Recap

- Removing items from a list:
  - .remove(item)
    - It does not return a value
  - del list[index]
  - .pop() or .pop(index)
    - It returns the removed value

Recap

- String to list:
  string.split(separator)
  - separator is optional; by default " "
  - It returns a list of strings

- List to string:
  string.join(list)
  - list must contain string items
  - It returns a string of all list items, separated by the string

Recap

- if statements are used to check a condition
  - if the condition is satisfied (True), it executes a certain code block
- elif statements are used to check for further conditions (optional)
  - if the condition is satisfied (True), it executes a certain code block
- else is executed if none of the above are True (optional)

- Syntax:

```
if condition_expression:
    # indented code block if condition is True
elif condition_expression:
    # indented code block if the if condition is False but elif condition is True
else:
    # indented code block if all above conditions are False
```

Breakout session II:
Selection Statement

Pass or fail (ex_5.2.py)

Write a program that requests the user to input a grade (in numerical form, e.g. 90). Then the program will display whether the class is PASSED or FAILED.

Use if and else statements to do this.

The table below can help you do the mapping.

Score	Result
>= 60	Passed
< 60	Failed

Test Score (ex_5.2.py)

Write a program that requests the user to input a grade (in numerical form, e.g. 90). Then your program will display the corresponding letter grade.

Use if, elif and else. The table below can help you do the mapping.

Test Score	Grade
90 and above	A
80 - 89	B
70 - 79	C
60 - 69	D
Below 60	F

Nested Decision Structure

- Decision structure can be nested inside each other:

```
if grade > 60:
    print ("You have passed")
    if grade < 70:
        print ("Your grade is D")
    elif grade < 80:
        print ("Your grade is C")
    elif grade < 90:
        print ("Your grade is B")
    else:
        print("Your grade is A")
else:
    print("You have failed the class")
```

Test Score	Grade
90 and above	A
80 - 89	B
70 - 79	C
60 - 69	D
Below 60	F

Logical Operators

Logical Operators

- Sometimes it is required to check several conditions at once
- Python has three logical operators to create complex Boolean expressions:
  - not: negate a Boolean expression
  - and: combines two Boolean expressions, both need to be True for the overall expression to evaluate to True
  - or: combines two Boolean expressions, only one needs to be True for the overall expression to evaluate True

not Operator

Used to negate the condition (the inverse of the condition):

```
guess = int(input("Make a guess: "))
if not guess < 7:
    print("Evaluated to True")
```

guess	Expression	Overall expression
-1		
10		

not Operator

Used to negate the condition (the inverse of the condition):

```
guess = int(input("Make a guess: "))
if not guess < 7:
    print("Evaluated to True")
```

guess	Expression	Overall expression
-1	True	False
10	False	True

and Operator

Check if **both** conditions are met:

```
guess = int(input("Make a guess: "))
if guess < 7 and guess > 0:
    print("Evaluated to True")
```

guess	Expressions	Overall expression
-1		
10		
5		
7		

and Operator

Check if both conditions are met:

```
guess = int(input("Make a guess: "))
if guess < 7 and guess > 0:
    print("Evaluated to True")
```

guess	Expressions	Overall expression
-1	True and False	False
10	False and True	False
5	True and True	True
7	False and True	False

or Operator

Check if one or both conditions are met:

```
guess = int(input("Make a guess: "))
if guess < 7 or guess > 0:
    print("Evaluated to True")
```

guess	Expressions	Overall expression
-1		
10		
5		

or Operator

Check if one or both conditions are met:

```
guess = int(input("Make a guess: "))
if guess < 7 or guess > 0:
    print("Evaluated to True")
```

guess	Expressions	Overall expression
-1	True or False	True
10	False or True	True
5	True or True	True

Order of Evaluation

- The order of evaluation from highest to lowest

Operator	Description
<, <=, >=, >, ==, !=, in	Comparison operators
not	boolean NOT
and	boolean AND
or	boolean OR

Breakout session I:
Logical expressions

Long Logical Expressions

Solve the following logical expressions on a piece of paper:

x = True
y = False
z = False

```
>>> x or y
>>> True
>>> not x and y
>>> False
>>> x or not y and z
>>> True
>>> not x or y or not y and x
>>> True
>>> not x and not y and not y and x or z
>>> False
```

Short-circuit Evaluation

- The and and or operators are evaluated using short-circuit evaluation
  - As soon as the truth value of the entire expression is determined, no further conditions are evaluated

```
donuts = 50
students = int(input("Number of students: "))
if students > 0 and donuts/students < 1:
    print("Insufficient donuts :(")
```

Chaining Comparison Operators

Comparison operators can be chained:

```
x = 2
if x > 1 and x < 3:
    print("x == 2")
```

Chained comparison operators:

```
if 1 < x < 3:
    print("x == 2")
```

Count-controlled Loop

Loops

- There are two kinds of loops:
  - A count-controlled loop (for loop)
  - A condition-controlled loop (while loop)

- A count-controlled loop iterates a specific number of times

The for Loop

- A for loop is used for iterating over a sequence

- A sequence can be:
  - string
  - list
  - range() sequence

- Syntax:

```
for item in sequence:
    #indented code block
```

The for Loop

- Syntax:

```
for item in sequence:
    #indented code block
```

- The for loop execution is as follows:
  - The first element in the sequence is assigned to the temporary variable item
  - The indented code block is executed
  - The loop repeats by assigning the next element in the sequence to item

- The for loop keeps repeating until the last element of the sequence is reached

Looping over Strings

```
for char in "Jon":
    print(char)
```

1st Iteration: `for char in "Jon":`
`print(char)` J

2nd Iteration: `for char in "Jon":`
`print(char)` o

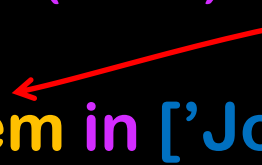
3rd Iteration: `for char in "Jon":`
`print(char)` n

Looping over Lists

```
for item in ['Jon', 'Sansa', 'Arya']:
    print(item)
```

1st Iteration: `for item in ['Jon', 'Sansa', 'Arya']:`
`print(item)` Jon

2nd Iteration: `for item in ['Jon', 'Sansa', 'Arya']:`
`print(item)` Sansa



3rd Iteration: `for item in ['Jon', 'Sansa', 'Arya']:`
`print(item)` Arya

The range() Function

- The range() function is used to create sequences of integers

  >>> range(start, end, step)

- Arguments:
  - start: first integer (inclusive) of the sequence (optional)
  - end: last integer (exclusive) of the sequence, starting from 0 (inclusive)
  - step: step size (optional)
    - default +1

- Returns a sequence ranging from start to end in steps of step

Examples: The range() Function

- Examples:

```
>>> range(5)          →   0, 1, 2, 3, 4

>>> range(5, 10)      →   5, 6, 7, 8, 9

>>> range(-1, 2)      →   -1, 0, 1

>>> range(0, 10, 2)   →   0, 2, 4, 6, 8
```

- Note: The output is NOT a list! It can be converted into a list using type casting:

```
>>> list(range(5))
[0, 1, 2, 3, 4]
```

The range() Function in Loops

- The range() function can be used as a sequence in a for loop

- Example:

```
for n in range(5):
    print(n)
```

```
0
1
2
3
4
```

Dynamically Creating Lists

- A for loop and the range() function can be used to dynamically fill lists during runtime

- Example:

```
my_list = []
for n in range(5):
    my_list.append(n)

print(my_list)        [0, 1, 2, 3, 4]
```

Breakout session II:
For loops

FIZZ BUZZ (ex_6.1.py)

Write a program that prints the numbers from 1 to 100

However, if the number is

- divisible by 3 print “FIZZ” instead of the number
- divisible by 5 print “BUZZ” instead of the number
- divisible by both 3 and 5, print “FIZZ BUZZ” instead of the number

Hint: Use the modulus operator %

Output:

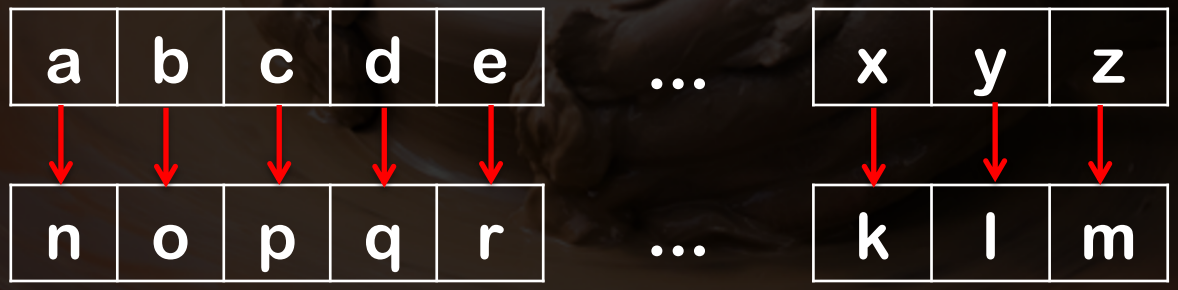
```
1
2
FIZZ
4
BUZZ
FIZZ
7
8
FIZZ
BUZZ
11
FIZZ
13
14
FIZZ BUZZ
16
17
...
```

```
message_list = [“Let’s”, “take”, “a”, “break!”]
for item in message_list:
    print(item, end= “ “)
```

Ciphered text using ROT13 (Caesar Cipher) (ex_6.2.py)

Julius Caesar used this ciphering technique to send messages of military significance to his army leaders.
http://en.wikipedia.org/wiki/ROT13

The cipher is a simple substitution cipher, that replaces each letter with the letter 13 letters after it in the alphabet.

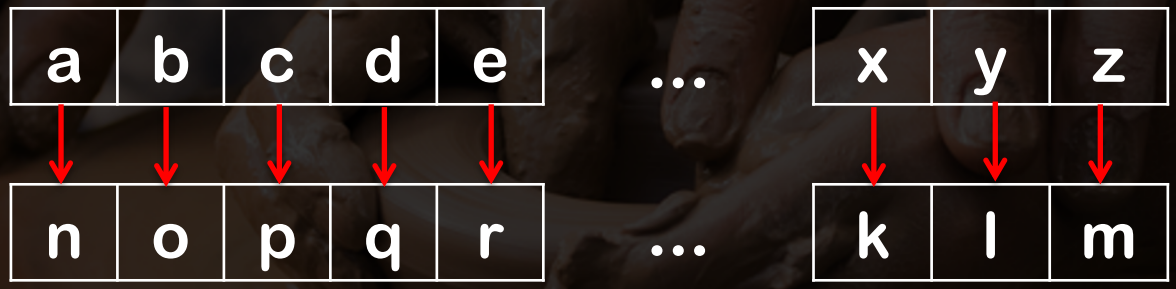


For example:

bayl -> only

Let's assume you have time travelled back to the year 60 BC. Luckily, you have your laptop with you with Python3 installed. You have intercepted the below message from Caesar. Write a program to decode the message. Your life might depend on it. ☺

lbh zhfg unir fbzr jvyq vzntvangvba. gvzr geniry? frevbhfyl!!! arirezvaq, yrgf gnxr n oernx abj. ohg ubyq ba, bar zber guvat. lbhe svefg ubzrjbex nffvtazrag jvyy or eryrnfrq gbqnl ba oevtugfcnpr. vg vf rapelcgrq jvgu ebg13. :)



You can find the encrypted message in ex_6.2.py on Brightspace.

Hint: Use a for loop to create the alphabet and the rotated alphabet. Also, use a for loop to loop over the encrypted message and decode it. Be aware of characters that are not in the alphabet!

Nested Loops

Nested Loops

- Nested for loops are a loop inside a loop

- Example:

```
message_list = [“Let’s”, “take”, “a”, “break!”]
for n in range(10):
    for item in message_list:
        print(item, end= “ “)
```

Nested Loops

- Example:

```
for outer_num in range(2):
    for inner_num in range(3):
        print(str(outer_num) + str(inner_num))
```

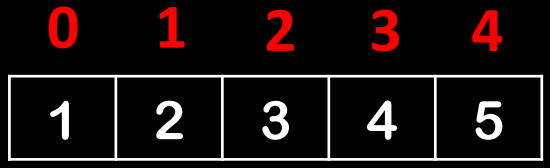
- Take pen an paper and think what the output of the code is!

Multi-dimensional Lists

Recap Lists

- Remember Lists?
  - List are mutable!
  - Defined by []
  - Lists can hold any data type
  - Elements are separated by commas:

numbers = [1, 2, 3, 4, 5]



List of Lists

- How about a list of lists?

  numbers = [[], [], [], []]

  numbers = [[1,2,3,4,5], [6,7,8,9,10],
  [11,12,13,14,15], [16,17,18,19,20]]

	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20

Accessing Elements of Lists

- Do you remember indexing of lists?
- numbers = [[1,2,3,4,5], [6,7,8,9,10], [11,12,13,14,15], [16,17,18,19,20]]

  – What is numbers[1] ?

  [6, 7, 8, 9, 10]

  – What is numbers[1][3] ?

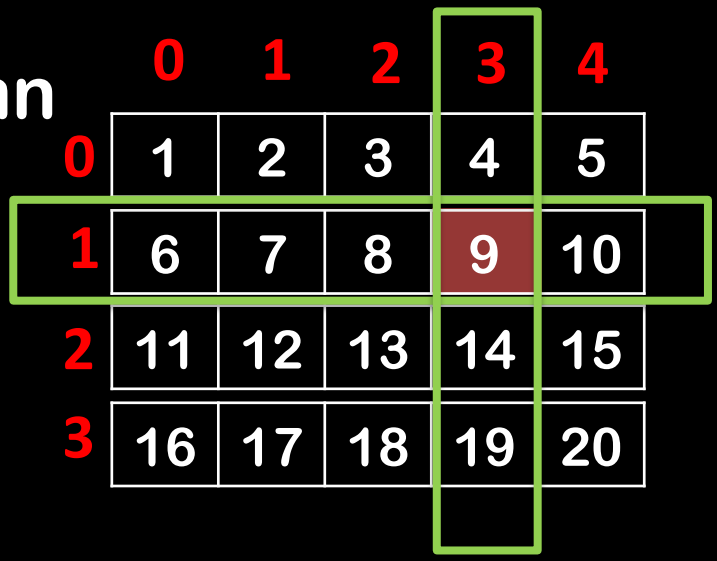
  9

	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20

Accessing Elements of a Lists

- numbers = [[1,2,3,4,5], [6,7,8,9,10], [11,12,13,14,15], [16,17,18,19,20]]

- numbers[1][3]
  - The first [] chooses a row
  - The second [] is the column



Multi-dimensional Lists

- How about a list of lists of lists?

```
numbers = [[[], []], [[], []], [[], []]]
numbers = [[[1, 2], [3, 4]],
           [[5, 6], [7, 8]],
           [[9, 10], [11, 12]]]
```

What are the dimensions of the list?

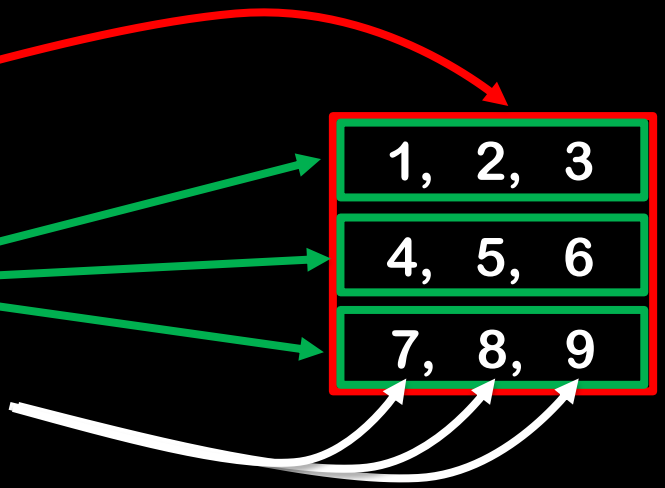
3 rows, 2 columns, 2 in depth

Creating 2D Lists

- How to create a 3x3 list that contains the numbers from 1 to 9?

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

- We need:
  - 1 outer list
  - 3 inner lists (rows)
    - values from 1-3, 4-6, 7-9

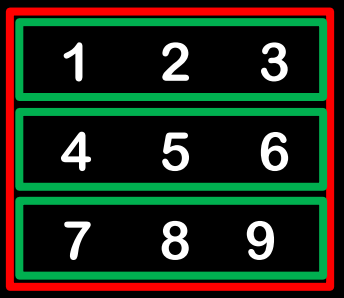


The Static Approach

```
outer_list = []
row_list_0 = []
row_list_0.append(1)
row_list_0.append(2)
row_list_0.append(3)
outer_list.append(row_list_0)

row_list_1 = []
row_list_1.append(4)
row_list_1.append(5)
row_list_1.append(6)
outer_list.append(row_list_1)

row_list_2 = []
row_list_2.append(7)
row_list_2.append(8)
row_list_2.append(9)
outer_list.append(row_list_2)
```



Dynamically Creating 2D Lists

```
cnt = 1
outer_list = []
for r in range(3):
        row_list = []
        for c in range(3):
                row_list.append(cnt)
                cnt = cnt + 1
        outer_list.append(row_list)

print(outer_list)
```

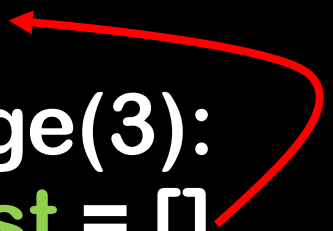
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]

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

How NOT to Create a 2D List

```
cnt = 1
outer_list = []

for r in range(3):
        row_list = []
        for c in range(3):
                row_list.append(cnt)
                cnt = cnt + 1
        outer_list.append(row_list)
```



What are the dimensions of the list?

How NOT to Create a 2D LSist

```
cnt = 1
outer_list = []

for r in range(3):
        row_list = []
        for c in range(3):
                row_list.append(cnt)
                cnt = cnt + 1
                outer_list.append(row_list)
```

What are the dimensions of the list?

Dynamically Printing 2D Lists

Option 1, using list indices:

```
for r in range(3):
        for c in range(3):
                print(outer_list[r][c], end=" ")
        print()
```

Note: You have to know the dimensions of the list!

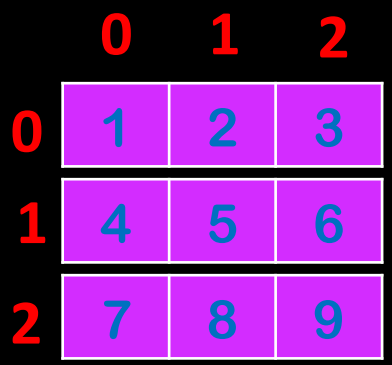
	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

Dynamically Printing 2D Lists

Option 2, using list items:

```
for row in outer_list:
        for item in row:
                print(item, end=" ")
        print()
```

Note: You do not have access to the indices of the list!



Next Class

- Condition-controlled Loops
- Lab on 2D-lists