CS65: Introduction to Computer Science

While Loop with If Statements
Print formatting

Md Alimoor Reza
Assistant Professor of Computer Science

Other Announcements

Emailing: Email me by stating your both names: English + Chinese. It's difficult to understand names from email address

Grading issues: For those who emailed me about grading issues, don't worry—I'll resolve them. For IS 44, if your automatic grade isn't correct, I'll manually override the score. So no need to stress or panic!

Section#1 Replacement Class for Oct 9th: I have another personal commitment on October 9, so the replacement class will be held on October 10 from 10:10 a.m. to 12:00 p.m.

Announcement

Lab #4 will be released on Thursday (09/07/2025) and it will be due on October 14th Tuesday for all sections

Assignment#1 (harder than Labs) will be also released on Thursday (09/07/2025) but it will be due October 24th Tuesday for all sections

if/elif/else selection statements

while loop

Text Reading

https://www.brianheinold.net/python/python_book.html#section_printing

"Printing"

Review from last lecture

- Invalid Inputs with While Loops
  - Hands-on Exercise
- Counting Loops
  - Hands-on Exercise
- Infinite Loops
  - Hands-on Exercise
- Accumulator with With Loops
  - Hands-on Exercise

Review: Infinite Loops

In all but rare cases, loops must contain within themselves a way to terminate

Something that makes the test condition false

If your loop does not have a way of stopping, it is called an *infinite loop*

```
counter = 0
while counter < 10:
    print("counter:",counter)
counter += 1
```

Review: Challenge #3

Prompt the user for an integer

Keep on dividing the number by 2 while the number is still greater than or equal to 1:

```
please enter a number: 64
64.0
32.0
16.0
8.0
4.0
2.0
1.0
```

```
please enter a number: 13
13.0
6.5
3.25
1.625
```

Review: Accumulator

An **accumulator** variable is a variable that keeps a running total of something.

```
1  # Alimoor Reza
2  # accumulator keeps adding something to a variable
3
4  total = 0
5  day = 1
6
7  while day <= 5:
8      amount = float(input("How much did you make today? "))
9      total = total + amount
10     day   = day + 1
11
12 print("you made a total of ", total)
```

Review: Challenge #4

- Write a loop that will execute 5 times
- Within the loop, get a float value from the user (the amount of rain that fell in a day)
- Use an accumulator to add up all of the rain that fell over the five days
- Outside of the loop, print out the sum of all of the rainfall

```
What is the rainfall? 2
What is the rainfall? 0
What is the rainfall? 1
What is the rainfall? 1.1
What is the rainfall? 3
There was a total of 7.1 inches of rain
```

Today's Plan

- If Statements within loops
  - Hands-on Exercise
- Print Formatting
  - Hands-on Exercise
- .lower() method
  - Hands-on Exercise
- Formatted Strings: f-strings
  - Hands-on Exercise
- Formatted Strings: .format() method
  - Hands-on Exercise

If statements within loops

Let's *trace* this code

"trace" – go through it line by line to determine how it works

Question: What is happening?

mystery_code.py

```
1  # Alimoor Reza
2  # if statement inside a while loop
3
4  num = 0
5  x   = 0
6
7  while num <= 5:
8
9      num   = num + 1
10     b     = int(input("please enter an integer between 0 and 100: "))
11
12     if b > x:
13         x = b
14 print("The mystery answer is ", x)
```

Challenge #1

step#1: Write the code that will loop 5 times

step#2: Within the loop, prompt the user to enter an integer that is between 0 and 100

step#3: After the loop, print out the *minimum* of the numbers entered

step#4: Use better variable names than the previous slide

Today's Plan

- If Statements within loops
  - Hands-on Exercise
- Print Formatting
  - Hands-on Exercise
- .lower() method
  - Hands-on Exercise
- Formatted Strings: f-strings
  - Hands-on Exercise
- Formatted Strings: .format() method
  - Hands-on Exercise

print

When you use the **`print`** function, it automatically outputs the *next* **`print`** statement on the following line of the output. For example, when you have

```
print("one")
print("two")
print("three")
```

What will be output?

print

When you use the **`print`** function, it automatically outputs the *next* **`print`** statement on the following line of the output. For example, when you have

```
print("one")
print("two")
print("three")
```

What will be output?

```
one
two
three
```

Print Formatting

If we didn't want the next print statement to output on the next line, we can pass in a second **argument** to the print function.

Passing in `end=""` will no longer make the next print statement continue on the next line.

- Thus,

Print Formatting

If we didn't want the next print statement to output on the next line, we can pass in a second **argument** to the print function.

Passing in **`end=""`** will no longer make the next print statement continue on the next line.

- Thus,

```
print("one", end="")
print("two", end="")
print("three", end="")
```

Print Formatting

If we didn't want the next print statement to output on the next line, we can pass in a second **argument** to the print function.

Passing in **`end=""`** will no longer make the next print statement continue on the next line.

- Thus,

```
print("one", end="")
print("two", end="")
print("three", end="")
```

```
onetwothree
```

Printing a blank line

If you want the output to contain a blank line, then print can be used without an **argument**

```
print("hello")
print()
print()
print("now I'm here")
```

Printing a blank line

If you want the output to contain a blank line, then print can be used without an **argument**

```
print("hello")
print()
print()
print("now I'm here")
```

```
hello


now I'm here
```

Challenge #2

Prompt the user for an integer

Print out a row of alternating X's and O's in which the number of characters equals the number input

```
Please enter an integer: 3
XOX
```

```
Please enter an integer: 8
XOXOXOXO
```

```
Please enter an integer: 45
XOXOXOXOXOXOXOXOXOXOXOXOXOXOXOXOXOXOXOXOXOXOX
```

(Extra) Challenging Exercises

Print out a 2D pattern like this:

XOXOXOXOXOXOXOXOXOXOXOXOXO

OXOXOXOXOXOXOXOXOXOXOXOXOX

XOXOXOXOXOXOXOXOXOXOXOXOXO

OXOXOXOXOXOXOXOXOXOXOXOXOX

Prompt the user for the number of rows and columns they would like.

Print Formatting `sep`

When *multiple* arguments are passed into the **print** function, they are by default separated by a space.

```
cost = 5.75
print("The cost of your item is $",cost)
```

Will output

```
The cost of your item is $ 5.75
```

Print Formatting `sep`

When *multiple* arguments are passed into the **print** function, they are by default separated by a space.

```
cost = 5.75
print("The cost of your item is $",cost)
```

Will output

```
The cost of your item is $ 5.75
```

Note the space between the $-*sign* and 5.75

Print Formatting

If you do not want the default space between arguments in a print statement, you can put an additional argument that specifies exactly what should be placed between items in the print statement

For example, nothing (the empty string) denoted `""`

Denoted `sep = ""`

Print Formatting

If you do not want the default space between arguments in a print statement, you can put an additional argument that specifies exactly what should be placed between items in the print statement

For example, nothing (the empty string) denoted “”

Denoted **sep = “”**

```
cost = 5.75
print("The cost of your item is $",cost, sep="")
```

Print Formatting

If you do not want the default space between arguments in a print statement, you can put an additional argument that specifies exactly what should be placed between items in the print statement

For example, nothing (the empty string) denoted “”

Denoted **sep = “”**

```
cost = 5.75
print("The cost of your item is $",cost, sep="")
```

```
The cost of your item is $5.75
```

Today's Plan

- If Statements within loops
  - Hands-on Exercise
- Print Formatting
  - Hands-on Exercise
- .lower() method
  - Hands-on Exercise
- Formatted Strings: f-strings
  - Hands-on Exercise
- Formatted Strings: .format() method
  - Hands-on Exercise

Another helpful function: `lower()`

```
answer = input ("It is hot outside? ")

if answer.lower() == 'yes':
    print("you bet it is")
else:
    print("it sure isn't")
```

lowerDemo.py

Challenge #3

The user can play a game. Each time they type in the word yes or yep (capitalization doesn't matter) then they earn $0.015. (1 and half cents)

As soon as they type in something other than yes or yep, then their turn is over.

Write the program that will implement this and also output how much they have earned (rounded to the nearest cent).

Pay close attention to the formatting:

```
would you like to play? yes
would you like to play? Yes
would you like to play? YEP
would you like to play? no
you earned $0.04
```

```
would you like to play? yes
would you like to play? YES
would you like to play? YEP
would you like to play? yep
would you like to play? yeS
would you like to play? no
you earned $0.07
```

Today's Plan

- If Statements within loops
  - Hands-on Exercise
- Print Formatting
  - Hands-on Exercise
- .lower() method
  - Hands-on Exercise
- Formatted Strings: f-strings
  - Hands-on Exercise
- Formatted Strings: .format() method
  - Hands-on Exercise

Formatted Strings

Python offers a powerful feature called f-strings (formatted string) to simplify printing out strings and variables

To use formatted string literals, begin a string with `f` or `F` before the opening quotation mark or triple quotation mark.

Inside this string, you can write a Python expression between { and } characters that can refer to variables

Formatted Strings Examples

```
cost = 5.75
print(f"The cost of your item is ${cost}")
```

Note the **f** before the quotation marks

Variable goes between { and }

Formatted Strings Examples

```
cost = 5.75
print(f"The cost of your item is ${cost}")
```

Note the **f** before the quotation marks

Variable goes between { and }

```
The cost of your item is $5.75
```

Formatted Strings Examples

```
cost = 5.75
item = "eggs"
print(f"The price of {item} is ${cost}")
```

Note the **f** before the quotation marks

Variable goes between { and }

```
The price of eggs is $5.75
```

Formatted Strings:
One more example

```
cost = 5.75
cost_per_egg = cost/12
print(f"The cost per egg is ${cost_per_egg}")
```

```
The cost per egg is $0.4791666666666667
```

Formatted Strings – format specifier

A format specifier is an optional part of a string formatting operation that tells Python how to display a value. It allows us to control aspects like:

- Number of decimal places
- Type of number representation (fixed-point, scientific notation, etc.)

Formatted Strings – format specifier

A format specifier is an optional part of a string formatting operation that tells Python how to display a value. It allows us to control aspects like:

- Number of decimal places
- Type of number representation (fixed-point, scientific notation, etc.)

```
print(f"The cost per egg is ${cost_per_egg:.3f}")
```

Formatted Strings – format specifier

A format specifier is an optional part of a string formatting operation that tells Python how to display a value. It allows us to control aspects like:

- Number of decimal places
- Type of number representation (fixed-point, scientific notation, etc.)

```
print(f"The cost per egg is ${cost_per_egg:.3f}")
```

Breaking Down `:.3f`

`:` starts the format specifier.

`.3` means round to three decimal places.

`f` stands for fixed-point notation (i.e., a regular decimal number).

Formatted Strings – format specifier

```
cost = 5.75
cost_per_egg = cost/12
print(f"The cost per egg is ${cost_per_egg}")

print(f"The cost per egg is ${cost_per_egg:.3f}")
```

Challenge #4

Prompt the user for a distance

Prompt the user for a time

Print out, “The car’s speed is …. miles per hour” rounded to two decimal places

```
enter a distance (in miles): 100
enter the time (in hours): 3
The car's speed is 33.33 miles per hour
```

String formating

```
# Alimoor Reza
# formatting examples

num1 = 12.3456
my_str1 = f"{num1:06.2f}"
print(my_str1)
```

String formating

```
# Alimoor Reza
# formatting examples

num1 = 12.3456
my_str1 = f"{num1:06.2f}"
print(my_str1)
```

Breaking Down `:06.3f`

: starts the format specifier.

06 allocate 6 columns and fill empty spaces with '0's

.2 means round to two decimal places.

f stands for fixed-point notation (i.e., a regular decimal number).

col_1	col_2	col_3	col_4	col_5	col_6
0	1	2	.	3	5

String formating

```
# Alimoor Reza
# formatting examples

num1 = 12.3456
my_str1 = f"{num1:06.2f}"
print(my_str1)
```

col1	col2	col3	col4	col5	col6
0	1	2	.	3	5

```
>>> %Run format_pattern2.py
  012.35
```

String formating

```
1  # Alimoor Reza
2  # formatting examples
3
4  my_str = "yo"
5  my_str1 = f"{my_str:>8}" # right alignment with '>' symbol
6  my_str2 = f"{my_str:<8}" # left alignment with '<' symbol
7  my_str3 = f"{my_str:^8}" # center alignment with '^' symbol
8
9  print(my_str1)
10 print(my_str2)
11 print(my_str3)
```

Breaking Down :>8

: starts the format specifier.

> means align characters to the right

8 means allocated 8 columns for the characters

col_1	col_2	col_3	col_4	col_5	col_6	col_7	col_8
						y	o
y	o						
			y	o			

String formating

```
1  # Alimoor Reza
2  # formatting examples
3
4  my_str = "yo"
5  my_str1 = f"{my_str:>8}" # right alignment with '>' symbol
6  my_str2 = f"{my_str:<8}" # left alignment with '<' symbol
7  my_str3 = f"{my_str:^8}" # center alignment with '^' symbol
8
9  print(my_str1)
10 print(my_str2)
11 print(my_str3)
```

col1	col2	col3	col4	col5	col6	col7	col8
						y	o
y	o						
			y	o			

```
>>> %Run format_pattern3.py
       yo
  yo
     yo
```

String formating

```
1  # Alimoor Reza
2  # formatting examples
3
4  my_str = "yo"
5  my_str4 = f"{my_str: ^8}" # center alignment + fill with '*' s
6  my_str5 = f"{my_str:*^8}" # center alignment + fill with '@' s
7  my_str6 = f"{my_str:@^8}" # center alignment + fill with '2' s
8
9  print(my_str4)
10 print(my_str5)
11 print(my_str6)
```

col_1	col_2	col_3	col_4	col_5	col_6	col_7	col_8
*	*	*	y	o	*	*	*
@	@	@	y	o	@	@	@
2	2	2	y	o	2	2	2

String formating

```
1  # Alimoor Reza
2  # formatting examples
3
4  my_str = "yo"
5  my_str4 = f"{my_str: ^8}" # center alignment + fill with '*' s
6  my_str5 = f"{my_str:*^8}" # center alignment + fill with '@' s
7  my_str6 = f"{my_str:@^8}" # center alignment + fill with '2' s
8
9  print(my_str4)
10 print(my_str5)
11 print(my_str6)
```

col_1	col_2	col_3	col_4	col_5	col_6	col_7	col_8
*	*	*	y	o	*	*	*
@	@	@	y	o	@	@	@
2	2	2	y	o	2	2	2

```
>>> %Run format_pattern4.py

    yo
***yo***
@@@yo@@@
```