CS65: Introduction to Computer Science

Print formatting
For Loops

Md Alimoor Reza
Assistant Professor of Computer Science

Announcement

Lab #4 will be released on Friday (09/10/2025) and it will be due on October 16th Thursday for all sections

Assignment#1 (harder than Labs) will be also released on Friday (09/10/2025) but it will be due October 24th Tuesday for all sections

- if/elif/else selection statements
- while loop
- for loop

Text Reading

https://python.swaroopch.com/control_flow.html

The For Loop

Review from last lecture

- 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

Review: 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)
```

Review: 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
```

Review: 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.

Review: 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

Review: 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
```

Review: 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

Review: 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

Review: 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
```

Review: 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
```

Review: 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.)

Review: 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).

Review: 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

Today's Plan

- More String Formatting
  - Hands-on Exercise
- For Loops
  - Hands on exercise 2,3,4: Describe code in English
- range()
  - Hands-on Exercise 5 (parts a-e)

Review: 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
```

Review: 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			

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

Review: 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

Review: 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@@@
```

Today's Plan

- More String Formatting
  - Hands-on Exercise
- For Loops
  - Hands on exercise 2,3,4: Describe code in English
- range()
  - Hands-on Exercise 5 (parts a-e)

The for loop: a count-controlled loop

Count-Controlled Loop: iterates a specific number of times.

Use a **`for`** statement (a ***`for loop`***) to write count-controlled loops

- Designed to work with a *sequence* of data items
  - Iterates once for each item in the sequence.
- General format:

```
for variable in [val1, val2, … , valn]:
      statements
```

Target variable: the variable which is the target of the assignment at the beginning of each iteration.

The for loop: a count-controlled loop

Count-Controlled Loop: iterates a specific number of times.

Use a **`for`** statement (a ***`for loop`***) to write count-controlled loops

- Designed to work with a *sequence* of data items
  - Iterates once for each item in the sequence.
- General format:

```
for variable in [val1, val2, … , valn]:
        statements
```

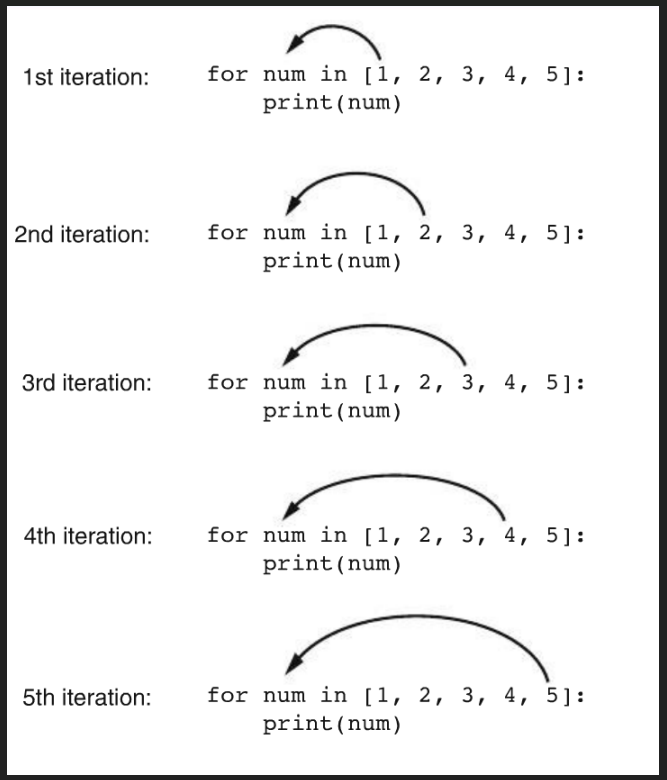
Target variable: the variable which is the target of the assignment at the beginning of each iteration.

Example for loop

```
for num in [1,2,3,4,5]:
    print(num)
```

Example For Loop

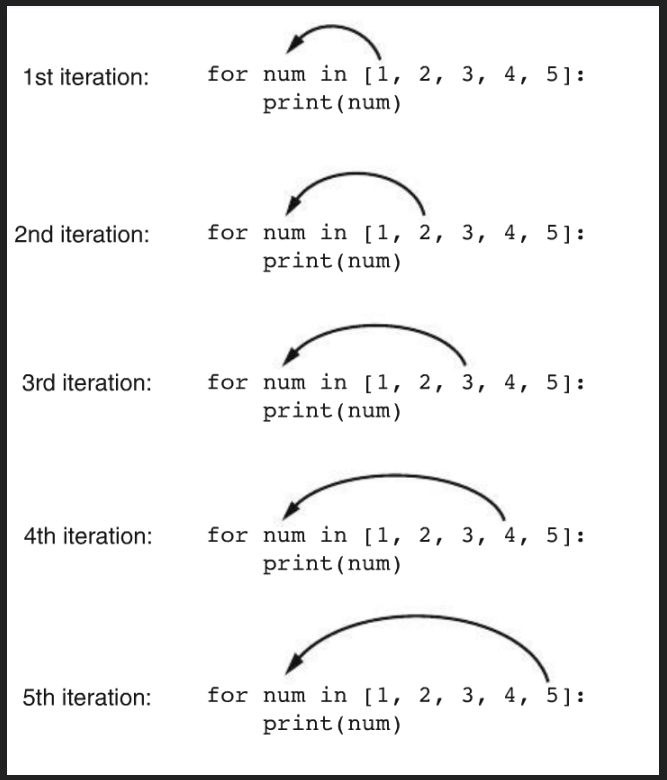
```
for num in [1,2,3,4,5]:
    print(num)
```



Example For Loop

```
for num in [1,2,3,4,5]:
    print(num)
```

```
1
2
3
4
5
```



For Loop Examples

```
# print odd numbers 1 thru 9
for val in [1,3,5,7,9]:
    print(val)
```

For Loop Examples

```
# print odd numbers 1 thru 9
for val in [1,3,5,7,9]:
    print(val)
```

```
1
3
5
7
9
```

For Loop Examples

```
for name in ["Harry", "Ron", "Hermione"]:
    print(name)
```

For Loop Examples

```
for name in ["Harry", "Ron", "Hermione"]:
    print(name)
```

```
Harry
Ron
Hermione
```

For Loop Example

forLoops.py

```
my_string = input("enter a string: ")

for ch in my_string:
    print(ch)
```

For Loop Example

forLoops.py

```
my_string = input("enter a string: ")

for ch in my_string:
    print(ch)
```

```
enter a string: asdf
a
s
d
f
```

Exercise #2

Describe in English what this code will do:

```
4  import random
5
6  total = 0
7  for day in [1,2,3,4,5]:
8      daily_stock = float(input("how much did your stock make:"))
9      total += daily_stock
10
11 print("you made a total of $", format(total,".2f"),sep="")
```

Exercise #2

Describe in English what this code will do:

```
3
4  import random
5
6  total = 0
7  for day in [1,2,3,4,5]:
8      daily_stock = float(input("how much did your stock make:"))
9      total += daily_stock
10
11 print("you made a total of $", format(total,".2f"),sep="")
```

DataCollection.py

Exercise #3

Describe in English what this code will do:

```
4  import random
5
6  total = 0
7  for day in [1,2,3,4,5]:
8      prompt_string = "how much did your stock make on day " + str(day) + ": "
9      daily_stock = float(input(prompt_string))
10     total += daily_stock
11
12 print("you made a total of $", format(total,".2f"),sep="")
13 print("on average, you made $", format(total/5,".2f")," per day",sep="")
```

DataCollectionPart2.py

Exercise #4

Describe in English what this code will do:

```
import random

number = random.randint(1,10)
for try_number in [1,2,3,4,5]:
    print("try", try_number, "--", end="")
    guess = int(input(" guess a number "))
    if guess == number:
        print("You got it!")
    else:
        print("Nope")
```

Exercise #4

guessingGame.py

Describe in English what this code will do:

```
import random

number = random.randint(1,10)
for try_number in [1,2,3,4,5]:
    print("try", try_number, "--", end="")
    guess = int(input(" guess a number "))
    if guess == number:
        print("You got it!")
    else:
        print("Nope")
```

Today's Plan

- More String Formatting
  - Hands-on Exercise
- For Loops
  - Hands on exercise 2,3,4: Describe code in English
- range()
  - Hands-on Exercise 5 (parts a-e)

`range()` function

The **`range`** function simplifies the process of writing a for loop

- `range` returns an *iterable* object
  - **Iterable**: contains a sequence of values that can be iterated (looped) over.

`range(5)` yields the same thing as [0,1,2,3,4]

Examples

forLoops2.py

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

```
0
1
2
3
4
```

Examples

forLoops2.py

```
for i in range(5):
    print("Hello World!")
```

```
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

Range()

Definition: An **argument** is an input into a function.

If you pass one argument to range, that argument is used as the ending limit of the number.

- Starting with 0

`range(5)` yields the same thing as [0,1,2,3,4]

Range()

If you pass two arguments to range, the first number is a starting value, and the second value is the ending limit.

`range(1,5)` yields the same things as [1,2,3,4]

Range()

By default, the range function produces a sequence of numbers that increases by 1.

If you pass three arguments to range:

- The first number is the starting value
- The second number is the ending limit (not included)
- The third number is the step value (increment)

`range(1,10,2)` yields the same thing as [1,3,5,7,9]

Exercise #5 – For Loops

a. Print out all integers between 1 and 200 (including 200)
b. Print out all even integers between 1 and 200 (including 200)
c. Print out all integers counting down from 50 to 1.
d. Print out all integers between 1 and 10, followed by the squares.

```
What is the limit? 10
What is the increment? 2
0
2
4
6
8
10
```

```
1 --- 1
2 --- 4
3 --- 9
4 --- 16
5 --- 25
6 --- 36
7 --- 49
8 --- 64
9 --- 81
10 --- 100
```

In-Class Participation Exercise

Describe a situation when you would use a while loop.

Describe a situation when you would use a for loop.