CMPT 120 Lectures 4-1 and 4-2

Prof. Nick Vincent

2023-09-25

Week 4

Welcome to Monday of Week 4!

Housekeeping

- New assignment will be posted tonight.
- Clarifying late submissions and fixing errors
- Housekeeping check-in?
  - How is Canvas working for you? How about Discord?
- Suggestions on effective question asking

Agenda for Lecture 4-1

- Practice questions + discuss
- Concept review: loops and why we use them
  - DRY: Don't repeat yourself
  - An "algorithm" for when to use loops
  - Changing things within a loop
- Concept review: types and converting
  - implict vs explicit type conversions
- Coding practice
- Break
- Start lecture 4-2

Practice Questions

- We'll start with some non-coding practice questions.
- No need to type any code here: just read the questions and do your best to answer.
- You may want to practice writing the answer in a blank text file or piece of paper, to simulate a test taking environment.

Question 1&2

What would the following code output?

Question 1

```
print("!.?hellothere".strip("!.h").upper())
```

Question 2

```
foods = ["Shakshuka", "Burrito", "Burger"]
print("Burger".upper().lower() in foods)
print("Burrito".lower().upper() in foods)
print("Taco" in foods)
```

Practice Question 3

What would the following code output?

```
a = "apple"
b = "banana"
c = "cherry"

if a + b in ["apple", "applebanana", "applecherry"]:
    print("Hello")
else:
    print("Goodbye")
```

Practice Question 4

```python
for i in ["0", "1", "2"]:
    print(i)
```

Practice Question 5

```python
is_a_good_day = False
response = input("Was today a good day?").lower()
if response in ["good", "great", "awesome"]:
    is_a_good_day = True
print(is_a_good_day)
```

What does this code print if user inputs:

- "I feel bad today"?
- "I feel good"?
- "GOOD"?
- "good"?

Loops

It's great for our well-being and for our software quality to avoid repeating ourself.

This has a led to a popular acronym in coding: DRY

Don't Repeat Yourself!

D.R.Y. Code

Read insightful Internet users' thoughts here (not sarcastic – it really can be insightful): - https://softwareengineering.stackexchange.com/questions/103233/why-is-dry-important

Some highlights:

- maintenance challenges
- readability
- extensibility
- easier testing

DRY in practice

If you find yourself writing

```
item1 = "soup"
item2 = "salad"
print('~~~' + item1 + '~~~')
print('~~~' + item2 + '~~~')
```

```
~~~soup~~~
~~~salad~~~
```

You may really want

```
for item in [item1, item2]:
    print('~~~' + item + '~~~')
```

```
~~~soup~~~
~~~salad~~~
```

Quick gotcha

I actually made this error while prepping these slides!

Can anyone catch it?

```
for item in [item1, item2]:
    print('~~~' + item1 + '~~~')
```

```
~~~soup~~~
~~~soup~~~
```

More motivation

It's even more motivating to use loops when your code gets complicated.

You may want to copy paste… but if what if something changes?

Imagine that you're performing a mathematical transformation… 100 times?

1000000 times?

More motivation

```
x = 5
a = complicated_function(x)
b = complicated_function(a) + other_variable
c = random_numer
d = a + b + c
print(d)
```

I don't want to write this 5 times, let alone 100!

Getting extreme with nested loops

Loops become even more powerful when we have multiple lists.

We'll see a variety of ways later on to handle many combinations of items.

As a quick example, say we want to print out a grid that looks like a game board.

```
for x in [1,2,3,4,5]:
    for y in [1,2,3,4,5]:
        print("()", end=" ")
    print()
```

```
() () () () ()
() () () () ()
() () () () ()
() () () () ()
() () () () ()
```

New argument in that last slide

- `print("()", end=" ")`
- `end` is a "keyword argument" we can "pass" to the `print()` function
- Useful to print without a newline character!

```
print("This", end=" ")
print("is all", end=" ")
print("on the same", end=" ")
print("line.", end="")
```

```
This is all on the same line.
```

Preview example: Times table

```python
for x in [1,2,3,4,5]:
    for y in [1,2,3,4,5]:
        answer = x * y
        print(str(answer).zfill(2), end=" ")
    print()
```

```
01 02 03 04 05
02 04 06 08 10
03 06 09 12 15
04 08 12 16 20
05 10 15 20 25
```

Mental model for loops

Whatever code is indented under your `for x in items:`

- will be run `n` times, where `n` is the length of `items`
- each time, the value of `x` will change to next item in `items`
- the indented code could itself include a loop (which will require more indendation!)

Type-related functions

- `int()`
- `float()`
- `str()`
- `bool()`
- `type()`: tells us the type of an object

Review: Type conversion

We can perform type conversions either implicitly (automatic) or explicitly (using type functions like `str()`).

- Implicit type conversion is a nice feature of Python, but can be a source of errors and gotchas!

- One classic “silent error”: concatenated two strings instead of adding two numbers
- One classic “not silent” error: adding a string and an int causes a crash

Implict Type Conversion examples

```python
result = 5 / 2  # Integer division that results in a float
print(result)  # Output will be 2.5
print(type(result))  # Output will be <class 'float'>
```

```
2.5
<class 'float'>
```

```python
result = 5 + 2.0  # Integer + Float = Float
print(result)  # Output will be 7.0
print(type(result))  # Output will be <class 'float'>
```

```
7.0
<class 'float'>
```

Do I need to memorize every implicit type conversion?

- In general, no!
- What’s more important is that when you’re coding, you’re careful to check your outputs and see if type conversion is causing problems.
- Some languages are very strict about types.
- Python is not, which often makes coding more pleasant – but we have to be cautious!

Explicit Type Conversions

```python
float_num = 3.5
int_num = int(float_num)  # Converting float to integer
print(int_num)  # Output will be 3
print(type(int_num))  # Output will be <class 'int'>
```

```
3
<class 'int'>
```

```python
int_num = 5
str_num = str(int_num)  # Converting integer to string
print(str_num)  # Output will be '5'
print(type(str_num))  # Output will be <class 'str'>
```

```
5
<class 'str'>
```

Note: hard to tell int and str apart when we print!

Explicit Type Conversions

```python
str_num = "3.5"
float_num = float(str_num)  # Converting string to float
print(float_num)  # Output will be 3.5
print(type(float_num))  # Output will be <class 'float'>
```

```
3.5
<class 'float'>
```

Note: this only works if our string is "compatible". Try `float("asd")`!

Coding practice 1

We want to write a countdown bot.

The bot should print the numbers 10 to 0 in reverse order.

Coding practice 2

Your boss wants you to write a countdown bot without a manually defined list.

Hint: your boss suggests using `range()` with three *arguments*.

Coding practice 3: Using a new method

Your boss wants you to write a countdown bot without a manually defined list and with only one argument passed to `range()`.

You should use `range(11)` and a new function, `reversed()`

Hint: `reversed([1, 2, 3]) gives you the list [3, 2, 1]`

Coding practice 4: code completion

- We want to build a simple restaurant recommender to help people in a small town pick between two restaurants
  - a restaurant with spicy meat dishes called "Spicy Meat Land"
  - a restaurant with a variety of non-spicy and vegetarian friendly options called "Many Options World".

Assume we get user input in two variables like so:

```
likes_spicy = input("Do you like spicy food?")
eats_meat = input('Do you eat meat?')
```

How should we print out a recommendation?

Break

See you in ten minutes!

Lecture 4-2 Agenda

- Discuss coding practice?
- Intro to recommendation systems

Questions about coding practice?

- Countdown bot
- Food suggestion

Intro to recommendation

Recommender systems – often called "RecSys" – are everywhere.

- Netflix
- Youtube
- Amazon
- TikTok (one huge recsys!)

We'll use recommender systems as our running example for a bit.

Concepts we'll continue to explore

We'll learn more about loops, integers and floats, performing calculations, using variables, operators, and more.

We'll also talk about files.

We'll learn the "accumulator" pattern

Very basic recsys

One easy way to recommend things is find the most popular thing - Which movie had the most sales or most reviews - Which website gets the most traffic - Which post got the most likes

Advanced recsys

Let's look at the history of every movie you've ever watched… and every movie everyone else has ever watched… and find all the connections to give you the perfect recommendation

In this course

If you want to, you can get really deep on the mathematics of recommendation! We'll just get a little taste in this course, however.

You might be interested to know however, that often times the "basic" approach – which is close to what we'll implement today, is actually quite good.

RecSys-like tasks in various domains

RecSys might seem mostly relevant to people who want "tech" jobs

But the core task of recsys: use what I like and what everybody else likes to match some content to me is broadly applicable

Polling and voting can be modelled as recsys problem ("Top 10 candidates for you!")

Recommendation can be made subject to constraints – suggest me something fun to do with minimal externalities

Think about how this might be relevant to you! We'll come back to it.

Ethical questions

- Does clicking or like something necessarily mean that I want to see more similar content?
- Can anyone think of example cases in which case the basic logic of recsys could cause problems?

Example: Most popular coffee

What's the most popular place to get coffee at SFU? - Starbucks? - Tim Hortons? - Renaissance Coffee?

(Seriously: I want to know!)

Approach: counting votes

We'll start by writing a Python script that will ask five different people for their favorite coffee.

Then we'll print out the number of votes each option received.

Counting

```
# This program should get input 5 times
# Each time, the program should ask users
# for report their favorite place to get coffee
# For this version, use a fixed set of options
```

Building up our code: part 1

```
# Store votes for starbucks in a variable.
# the variable starts at 0.
starbucks_votes = 0

# Get user input
# choice = input("What's your favorite place to get coffee?")

# placeholder answer for testing
choice = "starbucks"

# if the answers was starbucks, add one vote
if choice.lower() == "starbucks":
    starbucks_votes = starbucks_votes + 1

print(starbucks_votes)
```

```
1
```

Tactic: Hard code your input until the last minute

- If you have to type in an input every time, it may slow down your coding
- Try setting `choice = "something"`
- Finish all your code
- Then change it to `choice = input()` and verify it works

Next steps

How would we add another coffee shop?

Let's start with Tim Hortons.

Building up our code: part 2

```
# Store votes for starbucks in a variable.
# the variable starts at 0.
starbucks_votes = 0
tims_votes = 0
```

```
# Get user input
# choice = input("What's your favorite place to get coffee?")
choice = "starbucks"

# if the answers was starbucks, add one vote
if choice.lower() == "starbucks":
    starbucks_votes = starbucks_votes + 1
elif choice.lower() == "tim hortons":
    tims_votes = tims_votes + 1

print(starbucks_votes)
print(tims_votes)
```

```
1
0
```

Live code this together!

One answer

```
# This program should get input 5 times
# Each time, the program should ask users
# for report their favorite place to get coffee
# For this version, use a fixed set of options

# Initialize vote counts for each coffee place to 0
starbucks_votes = 0
tims_votes = 0
other_votes = 0

# Loop 5 times to get 5 different inputs
for i in range(5):
    # Display the options to the user
    print("Options: Starbucks, Tim Hortons, Other")

    # Get user input
    choice = input("What's your favorite place to get coffee? ")

    # Update vote counts based on user input
    if choice.lower() == "starbucks":
        starbucks_votes += 1
    elif choice.lower() == "tim hortons":
        tims_votes += 1
    else:
        other_votes += 1

# Display the final vote counts
print("Vote Counts:")
print("Starbucks:", starbucks_votes)
print("Tim Hortons:", tims_votes)
print("Other:", other_votes)
```

What if we want to use string concatenation

```
"Starbucks" + starbucks_votes
```

Does this work?

Using string concatenation: we have to convert

```
starbucks_votes = 15
tims_votes = 30
other_votes = 5
print("Starbucks:" + str(starbucks_votes))
print("Tim Hortons:" + str(tims_votes))
print("Other:" + str(other_votes))
```

```
Starbucks:15
Tim Hortons:30
Other:5
```

Percentages

How would we calculate the percentage of votes held by each options?

Percentages

To calculate the percentage of votes held by each option, we need to divide the number of votes for each option by the total number of votes and then multiply by 100

(Note: this is an algorithm: it's the algorithm for calculating a percentage)

Percentage example:

```python
# imagine we polled 50 people
starbucks_votes = 15
tims_votes = 30
other_votes = 5
# Calculate the total number of votes
total_votes = starbucks_votes + tims_votes + other_votes
# Calculate the percentage of votes for each option
if total_votes > 0:
    starbucks_percentage = (starbucks_votes / total_votes) * 100
    tims_percentage = (tims_votes / total_votes) * 100
    other_percentage = (other_votes / total_votes) * 100

    # Display the final vote counts and percentages
    print("Vote Counts and percentages:")
    print("Starbucks: " + str(starbucks_votes) + " (" + str(starbucks_votes / total_votes
    print("Tim Hortons: " + str(tims_votes) + " (" + str(tims_votes / total_votes * 100) +
    print("Other: " + str(other_votes) + " (" + str(other_votes / total_votes * 100) + ")"
else:
    print("No votes were cast.")
```

```
Vote Counts and percentages:
Starbucks: 15 (30.0)
Tim Hortons: 30 (60.0)
Other: 5 (10.0)
```

Tactic: work with a fixed input

Note that in the previous example, I once again set starbucks_votes, etc to fixed integers.

This way I can test really quickly without having to re-collect inputs.

Coding patterns used here

- Work with a fixed input
- Checking `if total_votes > 0:` for robustness
- Using parentheses to make code readable

Preview: f strings

- Our final print out with the vote counts AND percentages got a little clunky
- We broke our 100 characters per line rule!
- Python has a nice, but complicated way to handle this.
- We'll preview it now, but you don't have to use it unless you want to

How to use f strings

Recipe: - `f"Variable 1 is {variable_name1}"` - `f"Variable 2 with two decimal places is {variable_name2:.2f}"` - `{variable_name1}` says look for a variable named variable_name1 and fill in its values - `{variable_name:.2f}` says look for a variables named variable_name2 and print it as a float with 2 decimal places (.2 = two decimal places, f = float)

```
print("Vote Counts:")
print(f"Starbucks: {starbucks_votes} ({starbucks_percentage:.2f}%)")
print(f"Tim Hortons: {tims_votes} ({tims_percentage:.2f}%)")
print(f"Other: {other_votes} ({other_percentage:.2f}%)")
```

```
Vote Counts:
Starbucks: 15 (30.00%)
Tim Hortons: 30 (60.00%)
Other: 5 (10.00%)
```

Preview: the string `format` method

We can also define a string with `{}` or `{:.2f}` in it and call `.format()` to fill it in

```
print("Starbucks %: {:.2f})".format(starbucks_percentage))
```

```
Starbucks %: 30.00)
```

- Format string = no f in front of "”
- Confusing and easy to miss

Review snippet: Four ways to print so far

- using string concatenation
- using print() with multiple arguments
- using f strings
- using .format()

```python
name = "Alice"
# 1. Using string concatenation
print("Hello, " + name + "!")
# 2. Using print() with multiple arguments
print("Hello, ", name, "!", sep="")
# 3. Using f-strings
print(f"Hello, {name}!")
# 4. Using .format()
print("Hello, {}!".format(name))
```

```
Hello, Alice!
Hello, Alice!
Hello, Alice!
Hello, Alice!
```

You might also see old code using `%` operator to format strings. Not so common any more.

Counting our iterations

What if we want to tell each user how many iterations have passed so far?

Counting our iterations

```python
for i in range(5):
    print("You're user number " + str(i))
```

```
You're user number 0
You're user number 1
You're user number 2
You're user number 3
You're user number 4
```

Any problems with this?

Counting our iterations

```python
for i in range(5):
    print("You're user number " + str(i+1))
```

```
You're user number 1
You're user number 2
You're user number 3
You're user number 4
You're user number 5
```

```python
for i in range(1,6):
    print("You're user number " + str(i))
```

```
You're user number 1
You're user number 2
You're user number 3
You're user number 4
You're user number 5
```

Concept: zero-indexing

- Python is a zero indexed language
- There's a whole Wikipedia page about Zero-based counting:
- https://en.wikipedia.org/wiki/Zero-based_numbering
- There's a year zero in the Buddhist and Hindu calendars
- a zero floor in some buildings
- athlete number zero (PNW Damian Lillard fans might be familiar)

Some languages start counting at 1.

Review

- How do we translate “do the following things 100 times” into Python?
- How do we initialize a variable called `vote_count` with value 0?
- How do we add a vote to `vote_count`?

Review

- What are all the ways we know so far to change the types of variables?
- When would a type conversion fail and cause our code to crash?

Practice Questions, Round 2. Question 1

Look at the following code and answer the following questions.

```
item1 =  "1"
list1 = [item1, "2", "3"]

item4 = "four"
list2 = [item4, "five", "six"]

item7 = 7
list3 = [item7, 8, 9]
```

What is the data type of each of the following:

- item1
- item4
- item7
- list1

Question 2.

How could we rewrite this code with a manually defined list?

```
for i in [100, 101, 102]:
    print(i)

for j in [102, 101, 100]:
    print(i)
```

```
100
101
102
102
102
102
```

Question 3

Consider the following code. What will happen if you paste it into an empty .py file and try to run?

Can you make one edit to each line to make it print out “Your score is 20”?

```
#| echo: true
votes = 10
print("The total number of votes is " + score)
```

Question 4

Assume your friend wrote some code to get numberic input from a user. What might you do to process that input, just to be safe?

Question 5

What are two data types for numbers? Describe how they are different.

Question 6

What function could you use to

- get the number of elements in a list
- to iterate from some start number up until some stop number
- to flip the order of a list

Question 7

What will this code output?

```python
print(3 / 3)
```

```
1.0
```