CMPT 120 Week 3-3

Nick Vincent

2023-09-20

Agenda

Housekeeping

- New slides, who dis?
- Outstanding canvas issues

Review

- Concepts (high-level) vs. recipes (language rules)
- Review via code examples
- Special focus on Loops and Range (last lecture)

If time

- One more tool demo
- Ethics and applications
- Info about optional practice quiz

Housekeeping

- Trying out new slide format
  - Can share over GitHub
  - Some nice features for code highlighting
- Check in on Canvas quirks

Let's review

- Check out "learning goals" on Canvas
- We're going to rapid fire through them all right now

What we'll review

- Variable assignment
- Printing
- Processing strings
- Method chaining
- Boolean expressions
- `or` and `and`

What we'll review

- lists (`[...]`)
- user input (`input()`)
- Conditionals (`if True:`)
- Nested conditionals
- `in` for lists and strings
- `for` loops (also use `in`)
- `range()`
- Converting between data types ("5" -> 5)

Variable Assignment

```
# This is a comment
my_string = "this is a string"
multi_line_string = """
this string
is long
"""
my_string_single_quotes = 'single quotes'

my_int = 5
my_float = 5.5
```

Line Highlighting

```
print("I'm excited")
print("to highlight")
print("lines of code")
```

Printing

```
print("Hello")

print(
    "Print with",
    "commas"
    )
print(
    "Print " + "with concatenation!"
)
```

```
Hello
Print with commas
Print with concatenation!
```

Concepts: Assignment and data types

- `=` for assignment
  - variable assignment exists in every programming language
- `"quotes"` tell Python we have a string; numbers (like `x=5` or `x=5.5`) with no quotes tell Python we have an int or float
  - different data types exist in every language

Processing Strings

```
# Concatenating strings. Note the whitespace!
my_concatenated_string = "my " + "string"

# upper and lower
```

```
my_uppercase_string = "RAHHHH".lower()
my_lowercase_string = "rahhhh".upper()

# stripping whitespace
my_stripped_string = "   Hello   ".strip()

# stripping characters
my_stripped_string2 = "...Hello...".strip(".")
print(
    my_concatenated_string, my_uppercase_string,
    my_lowercase_string, my_stripped_string,
    my_stripped_string2
)
```

```
my string rahhhh RAHHHH Hello Hello
```

Method chaining

```
my_variable = "RAHHH  "
# note that applying string methods works
# on variables or "string literals"

print(
    my_variable.lower().upper().strip()
)
```

```
RAHHH
```

Boolean expression

```
# == and !=
print(
    2+2 == 4,
    "Nick" + "Vincent" == "NickVincent",
    2+2 != 3
)
# less than and greater than
print(
    2+2 > 3,
```

```
    2+2 < 5
)

# not
print(
    not 2+2 == 5,
    not True
)
```

```
True True True
True True
True False
```

`or` and `and`

```
first_input = True
second_input = False

print(
    first_input and second_input,
    first_input or second_input
)
```

```
False True
```

Preview: Chained booleans

- Python reads left to right, in general
- evaluate the `and` operations first
- evaluate the `or` operations second
- You can (and should) use parentheses or intermediate variables to make things readable
- Worry about this more in later classes (logic, discrete math)
- We'll see a strategy for this class in next slide

Preview: Chained booleans

```
A  = True
B = True
C = False
if A and B or C:
    print("What's going on here?")
# This is how Python treats it
if (A and B) or C:
    print("This is a bit more readable")

first_condition = A and B
second_condition = C
if first_condition or second_condition:
    print("That's easier to read!")
```

```
What's going on here?
This is a bit more readable
That's easier to read!
```

Lists

```
my_list = ["apple", "banana"]

# Can go many lines
my_list = [
    "apple",
    "banana"
]
```

In practice, we'll often populate our lists from external sources (file, spreadsheet, database).

User input

```
reply = input()
reply2 = input("Give me a reply")
input() # takes input but doesn't save it

# (we could put these into a list)
```

```
my_replies = [reply, reply2]
```

Importing random

```
import random
my_list = ["apple", "banana"]
random.choice(my_list)
```

```
'apple'
```

Recipe here: - `import name_of_module` - use `name_of_module.name_of_function` - random is the module, choice is the method

Many combos of `if`

```
print(
    "Pick an option: A, B, or C"
)
print("it's ok if it's lowercase and has whitespace")
user_pick = "a " # we could use input() here
user_pick_processed = user_pick.strip().upper()

if user_pick_processed == "A":
    print("A")
elif user_pick_processed == "B":
    print("B")
elif user_pick_processed == "C":
    print("C")
else:
    print("You didn't pick one of the options!")
```

```
Pick an option: A, B, or C
it's ok if it's lowercase and has whitespace
A
```

Many combos of if

- if
- if / else
- if / elif
- if / infinitely many elifs
- if / elif / else
- Important: every `if` starts a new block!

Nested if

- Can make any flowchart you can dream of
- Code should kinda look like a flowchart

```
first_variable, second_variable, third_variable = True, True, True
if first_variable:
    if second_variable:
        if third_variable:
            print("All three are true")
```

```
All three are true
```

Nested if

```
first_variable, second_variable, third_variable = True, True, False

if first_variable:
    if second_variable:
        if third_variable:
            print("true, true, true")
        else:
            print("true, true, false")
    else:
        if third_variable:
            print("true, false, true")
        else:
            print("true, false, false")
else:
    pass
```

```
true, true, false
```

In (for list inclusion)

```
my_list = ["apple", "banana"]

# Use `in` to see if a value appears as an entry of the list entries
print(
    "apple" in my_list,
    "kiki" in my_list
)
```

```
True False
```

In (for string inclusion)

```
# Use `in` to see if a string appears as "substring" in another string

print(
    "app" in "apple",
    "banana" in "apple"
)
```

```
True False
```

for loops

Recipe: `for variable_name_of_your_choice in my_list:`

```
#     do something n times
for fruit in my_list:
    print(fruit)

# does the same thing (I just named my variable x instead of fruit)
for x in my_list:
    print(x)

# I can define my list in the for loop if I like!
```

```
for x in [1,2,3]:
    print(x)
```

```
apple
banana
apple
banana
1
2
3
```

for loops with range()

```
# range(3) gives us 0,1,2 (but no 3)
# range(3,6) gives us 3,4,5
print(
    range(5)
)

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

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

index variable

- We might use `range(n)` to do something n times
- i is an "index variable"

```
# Goal: print "HELLO" 10 times
for i in range(10):
    print("HELLO")
```

```
HELLO
HELLO
HELLO
HELLO
HELLO
HELLO
HELLO
HELLO
HELLO
HELLO
```

range() with increments

```python
# Range
# range(0,10,2) goes in steps of 2
for number in range(0,10,2):
    print(number)
```

```
0
2
4
6
8
```

Preview: arguments

When we put multiple number separated by commas in range(…)

We're passing multiple "arguments" same as print(1, 2, 3)

```python
# 1 argument
range(10)
print('Hi')
# 2 arguments
range(0,10)
print('Hi', 'there')
# 3 arguments
range(0,10,2)
print('Hi', 'there', 'friend')
```

```
Hi
Hi there
Hi there friend
```

New Content (for lecture)

- str(...) and int(...)
- `str()` tries to turn something into a string
- `int()` tries to turn something into an int

int to str and str to int

int to str

```
x = 5

print(
    str(x),
    x + x,
    str(x) + str(x)
)
```

```
5 10 55
```

str to int

```
x = "5"
print(
    int(x),
    x + x,
    int(x) + int(x),
)
```

```
5 55 10
```

Concatenated vs. Addition

- Python will look at the data type to determine how "+" is interpreted
- If it's strings, concatenate
- If it's ints, add

Gotcha warning!

- Python will sometimes try to help you out by automatically converting things
- But not always

Try this:

```
# This doens't run!
str(5) + int(5)
```

How could this go wrong?

Booleans convert too

```
print(
    str(True),
    str(2+2 == 4),
    str(2+2 == 5),
    int(True),
    int(2+2 == 4),
    int(2+2 == 5),
)
```

```
True True False 1 1 0
```

Implicit type conversion is nice

```
if 1:
    print("Python converted 1 to True")
if 0:
    pass
else:
    print("Converted 0 to False")

if not 0:
    print("Converted 0 to False, then not False converted to True")

my_list = ["apple"]
if my_list:
    print("Python converted my_list to True")
```

```
my_list2 = []
if my_list2:
    pass
else:
    print("Python converted my_list2 to False")
```

```
Python converted 1 to True
Converted 0 to False
Converted 0 to False, then not False converted to True
Python converted my_list to True
Python converted my_list2 to False
```

Test it out

Let's take a minute – in your favorite REPL, try out as many combinations of conversions as you can. Report back on anything strange or unexpected!

Discuss concepts vs. recipes

Ethics and applications

- Concerns you have an are interested in?
- Applied examples
  - Public health context
  - Others?