Week 3-3

Nick Vincent

2023-09-20

Agenda

Housekeeping

- New slides, who dis?
- Outstanding canvas issues

Review

- Concepts (high-level) vs. spells (language rules)
- Walk through code of all 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
- lists
- user input
- Conditionals (if statements)
- Nested conditionals
- in for lists and strings
- for loops
- range()

New content

- Converting between data types ("5" -> 5)

Some variables we've seen

First, let's just review various ways to write strings and numbers (ints and floats).

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

```
my_string = "this is a string"
multi_line_string = """
this string
can go on for many lines
"""
my_string_single_quotes = 'single quotes'
my_int = 5
my_float = 5.5
```

Printing

```
print("Hello")

print(
    "Print with",
    "commas"
    )
print(
    "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 with no quotes tell Python we have an int or float
  - different data types exist in every language

Processing Strings

```
# Concatenating strings
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(".")
```

Method chaining

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

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

Boolean expression

```
# double equals
2+2 == 4
"Nick" + "Vincent" == "NickVincent"
# not equal
2+2 != 3
# less than and greater than
2+2 > 3
2+2 < 5
# != and not
2+2 != 5
not 2+2 == 5
```

`or` and `and`

```
first_input = True
second_input = False

first_input and second_input # False
first_input or second_input # 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

Preview: Chained booleans

```
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!")
```

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)
```

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 = input()
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!")
```

Many combos of if

- if
- if / else
- if / elif

- if / elif / else
- every `if` starts a new block!

Nested if

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

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

Nested if

```
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
```

In (for list inclusion)

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

In (for string inclusion)

```
"app" in "apple"
"banana" in "apple"
```

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)
```

for loops with range()

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

for number in range(3):
    print(number)
```

index variable

We might use `range(n)` to do something n times

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

range() with increments

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

Preview: arguments

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

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

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

New Content!

- 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
str(x)
x + x
str(x) + str(x)
```

str to int

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

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 try to help you out by automatically converting things

Try this:

```
str(5) + int(5)
```

How could this go wrong?

Booleans convert too

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

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")
```

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!