Fall 2022

Class 01 - Control Flow Practice

Today's Goals

1. Announcements

2. Practice and review control flow statements (while and if-then)

3. Learn about relative reassignment operators and elif statements

Tutoring!

- **Tutoring is your place for conceptual help at a personalized pace**, there is no time limit to interactions like Office Hours
- **Tuesday, Wednesday 4-7pm in SN Lobby**; schedule located at course.care
- You can have **1-on-1 interactions with TAs or visit with a group** of other students to go over similar concepts
- Great place to **review lecture material and in-class exercises, go over quizzes, or study** for upcoming quizzes
- **No exercise help** is offered through tutoring, visit Office Hours for all Exercise questions

Office Hours!

- Full schedule located at course.care
- **Office Hours is where to come when you face roadblocks on exercises**.
  - 15-minute meetings for 1-on-1 exercise help
- You can also ask conceptual questions, go over quiz, etc. *If there is not a queue of people waiting*, conceptual questions are less restricted on time.
- In the past week: 175 of you have made use of office hours and the average wait time has been 0.9 minute (with a standard deviation of 1.6 minutes)
- Come to Sitterson Hall lobby, submit a well articulated ticket on Course.Care, and we'll call you in as quickly as we can.

Practice Question Tracing Control Flow

Produce an environment diagram for the following code listing when the number input by the user is 3.

```
"""Tracing Loops Practice Question."""

i: int = int(input("Give a # > 0: "))
s: str = ""

while i > 0:
    s = s + "h"
    h: int = 0
    while h < i:
        s = s + "e"
        h = h + 1
    i = i - 1

print(s)
```

```python
"""Tracing Loops Practice Question."""

i: int = int(input("Give a # > 0: "))
s: str = ""

while i > 0:
    s = s + "h"
    h: int = 0
    while h < i:
        s = s + "e"
        h = h + 1
    i = i - 1

print(s)
```

Reassigning a variable relative to its current value

- This is a common pattern with many operators.

- When looping with a counter variable:

```
i = i + 2
# or
i = i - 1
```

- When adding some number to a total:

```
total_dollars = total_dollars + next_donation
```

- Building up a string with concatenation:

```
message_to_send = message_to_send + "!!!"
```

- Notice with long variable names this is a lot of redundancy to type!

Relative Reassignment Operators

When reassigning a variable relative to its current value, such as:

```
i = i + 1
```

The *addition reassignment operator* shorthand has the same effect:

```
i += 1
```

There are reassignment operators for other operators, as shown in the table.

Since you will use meaningfully descriptive variable names, this is a big improvement!

```
total_dollars = total_dollars + next_donation
# vs
total_dollars += next_donation
```

Before	After
`i = i + expr`	`i += expr`
`i = i - expr`	`i -= expr`
`i = i * expr`	`i *= expr`
`i = i / expr`	`i /= expr`
`i = i % expr`	`i %= expr`
`i = i // expr`	`i //= expr`
`i = i ** expr`	`i **= expr`

Warm-up question rewritten with relative reassignment operators.

Before...

```
"""Tracing Loops Practice Question."""

i: int = int(input("Give a # > 0: "))
s: str = ""

while i > 0:
    s = s + "h"
    h: int = 0
    while h < i:
        s = s + "e"
        h = h + 1
    i = i - 1

print(s)
```

After...

```
"""Tracing Loops Practice Question."""

i: int = int(input("Give a # > 0: "))
s: str = ""

while i > 0:
    s += "h"
    h: int = 0
    while h < i:
        s += "e"
        h += 1
    i -= 1

print(s)
```

Goal: Come up with one value of **x** where these two code listings would produce *different* output.

Discuss: Why? What is the difference between them?

```
if response == 0:
    print("Most definitely.")

if response == 1:
    print("Quite possibly.")

if response == 2:
    print("Ask again later.")
else:
    print("Nope.")
```

```
if response == 0:
    print("Most definitely.")
else:
    if response == 1:
        print("Quite possibly.")
    else:
        if response == 2:
            print("Ask again later.")
        else:
            print("Nope.")
```

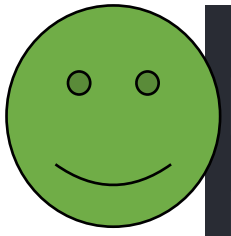
Nested if-else logic for only one of many paths...

- If your program has a point where control should flow in only one of many unique paths, you should express it with **nested if-else** statements rather than independent if-else statements.
- Why? As soon as the correct path is encountered, no additional conditions are evaluated. More efficient and avoids accidental logical errors

```
if response == 0:
    print("Most definitely.")

if response == 1:
    print("Quite possibly.")

if response == 2:
    print("Ask again later.")
else:
    print("Nope.")
```



```
if response == 0:
    print("Most definitely.")
else:
    if response == 1:
        print("Quite possibly.")
    else:
        if response == 2:
            print("Ask again later.")
        else:
            print("Nope.")
```

Eight Ball

Rewriting nested if-else statements with **elif**

- When you have many possible outcomes, deeply nested if-else-if-else statements grow quite cumbersome to write *and read*.
- Python (and most languages) have a shorthand way of expressing "else if" statements with equivalent semantics and abbreviated syntax...

Before...

```
if response == 0:
    print("Most definitely.")
else:
    if response == 1:
        print("Quite possibly.")
    else:
        if response == 2:
            print("Ask again later.")
        else:
            print("Nope.")
```

After...

```
if response == 0:
    print("Most definitely.")
elif response == 1:
    print("Quite possibly.")
elif response == 2:
    print("Ask again later.")
else:
    print("Nope.")
```

Search

Practice Question Tracing Control Flow

Imagine playing this game with a secret number of 18. At line 13, imagine correct inputs accordingly.

```
1   """Classic ordered searching algorithm."""
2
3   low: int = 1
4   high: int = 100
5
6   print("Think of a number between 1-100.")
7   input("Press enter to continue...")
8   playing: bool = True
9
10  while playing and low <= high:
11      guess: int = (high + low) // 2
12      print(str(guess) + "?")
13      result: str = input("Reply yes, higher, lower: ")
14      if result == "yes":
15          print("Woo!")
16          playing = False
17      elif result == "higher":
18          low = guess + 1
19      else:
20          high = guess - 1
```

```python
"""Classic ordered searching algorithm."""

low: int = 1
high: int = 100

print("Think of a number between 1-100.")
input("Press enter to continue...")
playing: bool = True

while playing and low <= high:
    guess: int = (high + low) // 2
    print(str(guess) + "?")
    result: str = input("Reply yes, higher, lower: ")
    if result == "yes":
        print("Woo!")
        playing = False
    elif result == "higher":
        low = guess + 1
    else:
        high = guess - 1
```

Notice the **elif** statement's use in this example

Before...

```
"""Classic ordered searching algorithm."""

low: int = 0
high: int = 100

print("Think of a number between 0-100.")
input("Press enter to continue...")
playing: bool = True

while playing and low <= high:
    guess: int = (high + low) // 2
    print(str(guess) + "?")
    result: str = input("Reply yes, higher, lower:")
    if result == "yes":
        print("Woo!")
        playing = False
    else:
        if result == "higher":
            low = guess + 1
        else:
            high = guess - 1
```

After...

```
"""Classic ordered searching algorithm."""

low: int = 1
high: int = 100

print("Think of a number between 1-100.")
input("Press enter to continue...")
playing: bool = True

while playing and low <= high:
    guess: int = (high + low) // 2
    print(str(guess) + "?")
    result: str = input("Reply yes, higher, lower: ")
    if result == "yes":
        print("Woo!")
        playing = False
    elif result == "higher":
        low = guess + 1
    else:
        high = guess - 1
```