Week 3 Review

While Loops

while: repeats code block under it while some conditional is true

```
print("meow")
print("meow")
print("meow")
# Not very efficient, use a loop

i = 1
while i <= 3:
    print("meow")
    i = i + 1
# Meows from i = 1, i = 2, i = 3 and stops.
```

Variations:

```
# Counting down from 3
i = 3
while i != 0:
    print("meow")
    i = i - 1

# Count up from 0
i = 0
while i < 3:
    print("meow")
    i = i + 1
```

A shorter way to increment / decrement a variable

```
i = i + 1
i += 1

i = i - 1
i -= 1

# same for other operators *, /, //, %, ||, &&, ...etc.
```

For Loops

For: run the loop once for each item in a sequence (a list, tuple, dictionary, string... etc)

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

# how many times do we meow?
```

range(start, stop, steps):

- function that returns a “range” type, which is essentially a list of numbers in sequence
- start: number to start at (inclusive), default 0
- end: number to end before (exclusive)
- step: interval between each number, default 1

```
range(3)          # [0, 1, 2]
range(3, 7)       # [3, 4, 5, 6]
range(0, 10, 2)   # [0, 2, 4, 6, 8]
range(10, 5, -1)  # [10, 9, 8, 7, 6]
```

```
for i in range(3):
    print("meow")

# how many times do we meow?
```

```
for _ in range(3):
    print("meow")
```

```
print("meow\n" * 3, end="")
```

Continue / Break

Sometimes we want to have a while loop that keeps looping until we specifically tell it to stop.

i.e: How do we keep asking for user input until we get an input we want?

```
while True:
    n = int(input("What's n? "))
    if n <= 0:
        continue
    else:
        break

for _ in range(n):
    print("meow")
```

continue:

- If code reaches `continue`, ignore everything left in the current iteration of the loop, and continue to the next iteration of the loop

break:

- If the code reaches `break`, exit the loop immediately
- If the loop is inside a function, we can also exit the loop with `return` (exit the loop AND function immediately)
- CAUTION: If you use `while True` , always make sure the loop can exit! Otherwise it will run forever

```
while True:
    n = int(input("What's n? "))
    if n > 0:
        break

for _ in range(n):
    print("meow")
```

Lists

List: a collection of objects; each object can be referenced with its index (starting from zero)

Zero-indexed: the first item has index 0, 2nd item has index 1 ... so on.

```
# A list of ints
my_list = [3, 1, 0, 2, 4]
print(my_list[0])           # prints 3
print(my_list[1])           # prints 1
print(my_list[2])           # prints 0
print(my_list[3])           # prints 2
```

Lists can contain other data types too

```
# A list of strings
students = ["Hermione", "Harry", "Ron"]
print(students[0])
```

```
print(students[1])
print(students[2])

# Print everything in the list
for student in students:
    print(student)

# Print the index (+ 1 since humans count from 1 instead of
0)
# Print the students name
for i in range(len(students)):
    print(i + 1, students[i])
```

len(): function that returns the number of items in a collection

Dictionary

dictionary:

- aka. “hash table”, “hash map”
- maps a collection of keys to values. We call these “key value pairs”
- Think of it like looking up a word in a dictionary to find information about the word
- Keys can be strings, integers, or even custom objects, but it must be a “hashable” type.
    - i.e. Lists are NOT hashable, so lists cannot be used as a key → use tuples instead!

```
students = {
    "Hermione": "Gryffindor",
    "Harry": "Gryffindor",
    "Ron": "Gryffindor",
```

```
    "Draco": "Slytherin",
}

print(students["Hermione"])
print(students["Harry"])
print(students["Ron"])
print(students["Draco"])

# Prints every student name (key) and house (value) in dictio
nary
for student in students:
    print(student, students[student], sep=", ")
```

```
students = [
    {"name": "Hermione", "house": "Gryffindor", "patronus":
"Otter"},
    {"name": "Harry", "house": "Gryffindor", "patronus": "Sta
g"},
    {"name": "Ron", "house": "Gryffindor", "patronus": "Jack
Russell terrier"},
    {"name": "Draco", "house": "Slytherin", "patronus": Non
e},
]

for student in students:
    print(student["name"], student["house"], student["patronu
s"], sep=", ")
```

Hashable

An object is “**hashable**” if some function can convert it into a unique value

- Imagine a library filled with books - how do we know where to find the book we want?

- We give each book a “code”, so that the librarian can look up the code and know where to find the book. The same book always maps to the same code, so we can easily find the same book again later.
- **hashing** is the process of assigning each object its lookup code

Mutable/Immutable

An object is **mutable** if its values can be changed in its lifetime in a program

- Mutable types: lists, dictionaries, sets, custom objects (usually)

An object is **immutable** if its values can’t be changed

- Immutable types: int, float, booleans, strings, tuples

```
my_list = ["a", "b", "c"]
my_list[0] = "not a anymore"

# my_list = ["not a anymore", "b", c"]
```

```
my_tuple = ("a", "b", "c")
print(my_tuple[0])      # prints "a"

my_tuple[0] = "not a anymore"
# this is not allowed!
# TypeError: 'tuple' object does not support item assignment
```

Tuples: a collection of objects that can be accessed by index (similar to lists), except it is immutable

- Why would we want a list that we can’t edit?
  - **immutable** makes tuples **hashable**! So tuples can be used as dictionary keys.
- More on tuples later...