Python (notes from Skillcrush classes)

- Variables - containers that hold the info you use in your code
  - Naming:
    - lower case - must start w/ letter or underscore & lower case is best practice
    - underscore between words - can't have spaces in variable names & underscores are preffered so you can distinguish words easily
    - descriptive - name your variable something that describes the purpose of the variable

  - Reserved words:
    * Cannot be used as variable names as these are keywords in Python used to identify other coding elements & perform operations

and	del	from	None	True
as	elif	global	nonlocal	try
assert	else	if	not	while
break	except	import	or	with
class	False	in	pass	yield
continue	finally	is	raise	
def	for	lambda	return	

  - Data Types: strings, integers, floats
  - Arethmetic Operators: (math) +, -, *, /
    - Concatenation: use + with strings to combine them

```
1 welcome_message = "welcome to the app, "
2 visitor = "Makayla!"
3
4 print(welcome_message + visitor)
```

```
>> Welcome to the app Makayla!
```

    - Repetition Operator: use * to print same thing multiple times

```
1 food_order = "pizza "
2
3 print(food_order * 3)
```

```
>> pizza pizza pizza
```

  - Comments: used to clarify code & leave other important info in program

```
#this is a comment
```

Control Structures & Booleans

Control Structures - used to first test a condition, then instruct the code to perform an action or to stop running.

Comparison operators - used to compare 2 or more values

`<` , `>` , `==` , `<=` , `>=` , `!=`

If Statements - used to test conditions + then make decisions to run a piece of code or not

- **elif** - else if
- **else** - last piece - kind of a last resort or default case

```
1  tacos = 0
2
3  if tacos == 0:
4      print("We need more tacos!")
5  elif tacos == 1:
6      print("Only one taco left. Time to panic!")
7  elif tacos == 2:
8      print("Two tacos left. Time to order more?")
9  else:
10     print("We have all the tacos!")
```

>> We have all the tacos!

Booleans - data w/ only 2 possible values: True or False; Always Cap!
Boolean Operators: and , or , not

Try + Except - keywords used to handle unexpected or incorrect data types or values from user input

- **try** - will test whether the user inputs a specified data type or specific value. if not correct, an error will occur.
- **except** - is triggered to print a message about the error

```
1  age = input("What is your age?: ")
2  try:
3      int(age)
4  except:
5      print("Your answer wasn't a number.")
```

len function will count # of chars in a string

Functions - blocks of code you can reuse

Building Functions:
```
def log_out(username):
    goodbye = "Thanks for visiting, " + username + "!"
    return goodbye
```

Modules - files of helpful code built by other developers that you can easily incorporate in your own projects
+ must import modules in order to use them:

ex:
```
import dancing_cat_module
import turtle as turtle
```

Lists - groups of different values; value stored in a list is called an **element**; can have multiple data types

writing lists:
```
female_superheros = ["Wonder Woman", "Cat Woman", "Flashgirl", "Storm"]
prices = [1.56, 2.86, 3.43]
```

add element:
```
prices.append(2.50)      -> adds to end
prices.insert(1, 4.27)   -> adds to index 1
```

remove element:
```
prices.pop()     -> removes last element
prices.pop(2)    -> removes 2nd index
```

selecting elements:
```
prices[3]        -> selects element at index 3
```
to reference more than 1 element at a time: (slicing list)
```
prices[1:3]   -> selects elements 1 & 2 but NOT 3
prices[:3]    -> selects start of list upto 3rd index
prices[3:]    -> selects end of list from 3rd to end
```

Loops - statements that allow you to repeat same code mult. times

for loops `for friends in invites:` repeat for each element inside of "invites"; does not matter what you name elements

`for i in range(0,6):` will run 6 times

while loop `while invites is < 10:` will run code until invites = 10 or more

Dictionaries

dictionary - data structure that lets you store groups of values; they have of data stored as pairs called key-value pairs

keys - unique elements in dictionary that are connected to a data value

values - data that connect to each key

Write a dictionary:

```
blairs_boutique = {
    "5635" : "Pink Power Purse",
    "2857" : "Pearl Clutch",
    "8503" : "Sassy Leopard Purse"
}
```

strings need quotes, ints + floats do not

keys ↑ separated by colon ↑ values

Get value from dict: `blairs_boutique.get("8503")` → returns value at 8503

add to dict: `blairs_boutique["5261"] = "Crystal Earrings"` → adds key-value pair 5261

remove values from dict: `blairs_boutique.pop("8503")` → removes key-value pair 8503

prints list of key-value pairs:

```
for key, value in blairs_boutique.items():
    print(key + " => " + value)
```

>> 2857 => Pearl Clutch ...

Classes

(OOP)

Object-oriented programming - type of coding where data structures & functions are combined

Class - blueprint for creating new Python objects
Object - data structure, like variable or list or dictionary

Steps to Write a Class

1. Define Class
2. Set up Class
3. Create properties for the class
4. Add functions + variables
5. Run the Class

1. Define a Class - Class names are always capitalized
   ```
   Class Song:
   ```

2. Set up a Class - Start by using the Constructor Function which builds the class & passes any relavent arguments
   ```
   def __init__(self, title, length):
   ```
   * self is shorthand for the Song class; think of it like Song class referring to itself

3. Create Properties for a class - Property is like a variable in a class w/ a value + name
   ```
   self.title = title
   self.length = length
   ```
   * self.title indicates you want to assign the title property to title arg set up in const.
   * dot b/n self + title is called dot notation + is how you add properties to a class

4. Add Functions + Variables

   Functions - create print function w/ info about the track
   ```
   def track(self):
       print("The song" + self.title + " is" + self.length + " long.")
   ```

   Variables - create variable to hold info about a single song outside the class
   ```
   song1 = Song("MMM Bop", "4:29")
   ```

Setters + Getters - to update incorrect info

setter - used to update property

getter - used to retrieve property

setter
```
def set_title(self, new_title):
    self.title = new_title
```

getter
```
def get_title(self):
    return self.title
```

5. Run the Class

use the print() function to see info

Whole Class: (example)

```
class Song:
    def __init__(self, title, length):
        self.title = title
        self.length = length

    def track(self):
        print("The song " + self.title + " is " +
        self.length + " long.")

song1 = Song("MMMBop", "4:29")
print(song1.track())
```

>> The song MMMBop is 4:29 long.

Working With Files in Python

File processing - system used to create, save, & retrieve data in files

"r" read
"w" write
"a" append

} modes which specify what will happen when the file is accessed in Python

Append text:

1st assign a variable (skills) then use open() function then write name of file & use "a"
2nd use write() function to add to file
3rd make sure to close the file

```
skills = open("python_skills.txt", "a")
skills.write("File processing")
skills.close()
```