1. Recall last time: we spoke about expressions:
    a. <object><operator><object>
        i. The following are types of expression:
2. Operations with ints/floats/doubles
    a. Given that i and k are both either: int/double/float
        i. i+j
        ii. i-j
        iii. i*j
        iv. i/j
        v. i%j
            1. the remainder when you divide i by j
        vi. i**j
            1. i raised to the power of j
3. Operator precedence
    a. Use parenthesis to add priority
        i. If you want some operation done first, use ()
    b. Otherwise, the precedence is:
        i. **
        ii. *
        iii. /
        iv. +
        v. –
    c. The above operations are executed from left to right!
4. Assignment:
    a. X=42;
        i. We are assigning the value of 42 to the variable X.
        ii. This effectively binds the name to the value, which is stored in memory
        iii. You can use the value, or obtain it by using the variable assigned to that value,
            1. In this case- X
    b. Much easier to give the name of X, rather than indicating 42 every time.
5. You can always re-bind variable names by making new assignment statements.
    a. Example:

```
X=42
X=X+100
```

6. Strings:
    a. This is an object type that consists of letters, spaces, digits, special characters.
    b. In python- you can indicate string by placing the string in either “” or ‘’
        i. Example:

```
theAnswer= “I can’t really say”
```

    c. You can do operations on strings:
        i. lunch= “hotdog” * 3;
    d. You can concatenate strings:
        i. answer= “42”
        ii. theAnswer=”The answer to everything is always”
        iii. fullAnswer=theAnswer+ “ ” + answer

e. the output of above:
    i. The answer to everything is always 42

7. Printing: Output
    a. Use print() to output to the screen
    b. You can print variables, but for a nice output, convert it to a string.
        i. X=42
        ii. print(X)
        iii. X_str=str(42)
        iv. print(X_str)
8. Input:
    a. Use this function: input("type some text here")
        i. To allow the user to input something into the program
        ii. First, your string: <type some text here> will print
        iii. Then the user will be allowed to enter something
        iv. You can also bind what the user entered into the program as follows:
            1. user_input=input("Please enter your name")
            2. print(user_input)
        v. KEEP IN MIND!!!
            1. Input gives you a string, so anything entered in using input will be bound as a string.
            2. If you want a number, you have to make sure you cast it to a number
                a. user_num=int(input("Please enter your favorite number"))
9. Comparison Operators:

```
1  i > j
2  i < j
3  i >= j
4  i <= j
5  i == j
6  i != j
```

    a. 1-4 check the values of i and j, and return true or false depending on the values and the operator used.
    b. 5-6 are equality operators:
        i. i==j
            1. true if i and j are equal.
        ii. i!=j
            1. true if I and j are not equal
10. Logical operators:
    a. not p
    b. p and q
    c. p or q

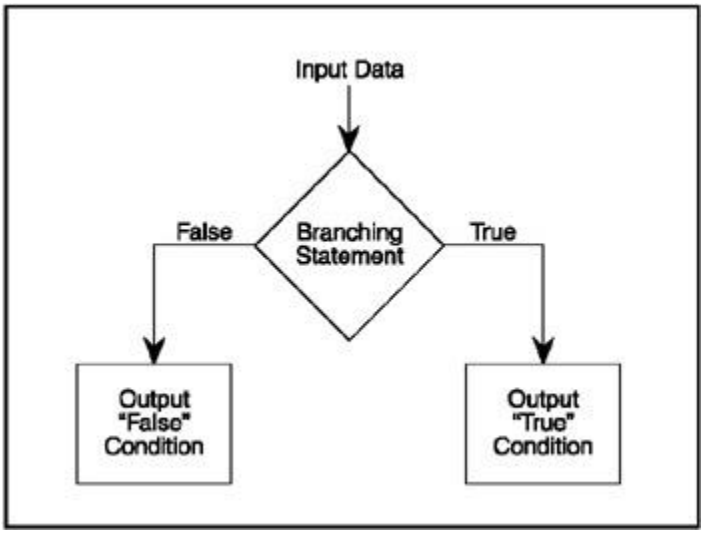
P	Q	P and q	P or q
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F

11. examples:

a.
```
>>> coder=True;
>>> hasALife=False
>>> both= coder and hasALife
>>> print(both)
False
```

b.
```
>>> lunchTime=2
>>> workTime=5
>>> print(lunchTime>workTime)
False
>>> print(lunchTime<workTime)
True
```

12. Control Flow:
    a. Branching:
        i. If … then do this
        ii. Else … do that

b.



```
salary=100
if(salary>1000000):
  print("Dude, you're rich!")
else:
  print("Keep trying")
```

```
Keep trying
```

==========

```
salary=100
if(salary>1000000):
  print("Dude, you're rich!")
elif (salary> 100000):
  print("Almost there!")
elif(salary> 10000):
  print("You can probably buy a new car")
elif(salary>1000):
  print("Hey, it's something")
else:
  print("Get job at google.")
```

```
Get job at google.
```

==========

```
hacker=True
if(hacker):
  print("Win all the wifi passwords")
else:
  print("Insecure starbucks wifi it is")
```

```
Win all the wifi passwords
```

13. Indentation in python:
    a. This is crucial!
    b. Anything indented belongs to the scope of what is not indented.
        i. Ex:

```
salary=100
if(salary>1000000):
  print("Dude, you're rich!")
elif (salary> 100000):
  print("Almost there!")
elif(salary> 10000):
  print("You can probably buy a new car")
elif(salary>1000):
  print("Hey, it's something")
else:
  print("Get job at google.")
```

ii.

iii. The first print statement belongs to the if stmt

iv. The second print statement belongs to the first elif statement.

c. Indents specify scope:

i. Scope- A scope defines the visibility of a name within a block. If a local variable is defined in a block, its scope includes that block.

ii. Also, if you enter an indented block, then you can execute those commands.

```
x=int(input("Enter in a number: "))
y=int(input("Enter in a number: "))

if(x==y):
  print("The two numbers are equal")
  if(y==0):
    print("Both numbers are zero")
else:
  print("The numbers are not equal")
  if(x>y):
    print("x is greater than y")
  else:
    print("y is greater than x")
```

iii. 4

14. Control flow: loops

a. Sentinel loop:

i. Loops that are continue until some special value is entered, in which case the loop terminates

b. Counter loop:

i. Loops that run up until some predefined number.

c. Conditional loops:

i. A mix of sentinel and counter,

1. Unknown how many times it will repeat, like sentinel
2. Like counter loops, they terminate as a result of calculations instead of based on user input.

15. Counter loops:

a. For loop:

b.
```
for counter in [1,2,3,4,5]: #[] in brackets, is an array
  print ("I am a counting loop")
```
c.
```
I am a counting loop
I am a counting loop
I am a counting loop
I am a counting loop
I am a counting loop
```
d. For loop:
   1. for- keyword, indicates a for loop
   2. counter- a new variable
   3. in – keyword, the variable counter is set to 1, goes up to 5.
   4. Everything indented under the for loop is executed five times.

   ii. for <variable> in range(<some_num>):
   iii. <expression>
   iv. <expression>
   1. The variable takes on a value (usually the first one from the range), and then executes the body of the loop
   2. Then the value is incremented to the next value in the range, usually plus one.
   3. Repeat until the value hits the upper bound of the range, then stop.

   v. More convenient for loop:

1.
```
for i in range(5):
  print("I am a counting loop")
```
```
I am a counting loop
I am a counting loop
I am a counting loop
I am a counting loop
I am a counting loop
```
2. Using the range function, which gives you the numbers: 0,1,2,3,4 (starting at 0, up til and not including the number you indicated-5, in this case)

3.
```
for i in range(5):
  print(i)
```
```
0
1
2
3
4
```
e. Counter while loop:
   i. Format:
      1. while <condition>:
      2. <expression>
      3. <expression> …
   ii. If the condition is true,
      1. Execute everything within the indented block
   iii. Repeat until the condition is false
   iv. Same as saying: do everything in the block while the condition remains true, when false-stop

f.

```
i=0
while(i<5):
  print("A counter while loop")
  i+=1
```

```
A counter while loop
A counter while loop
A counter while loop
A counter while loop
A counter while loop
```

i. Counter because it executes the loop a predetermined number of times, which is maintained by incrementing the variable i by one every time the loop executes.

16. Sentinel loops:
    a. While loop:
        i.

```
coffee=int(input("How many cups of coffee did you drink?"))
while(coffee<3):
  print("You need more coffee")
  coffee=int(input("How many cups of coffee did you drink?"))
print("You should be awake now")
```

```
How many cups of coffee did you drink? 0
You need more coffee
How many cups of coffee did you drink? 1
You need more coffee
How many cups of coffee did you drink? 2
You need more coffee
How many cups of coffee did you drink? 2
You need more coffee
How many cups of coffee did you drink? 3
You should be awake now
```

        ii. While sentinel:
            1. You have to initiate the variable coffee before the while loop runs, otherwise the variable coffee is filled with garbage that will not be interpreted correctly.
            2. You have to allow the user to change the variable coffee, otherwise it will always stay the same value, creating an infinite loop.

17. Break and continue:
    a. break
        i. when this cmd is executed, the control flow immediately breaks out of whatever loop it is in.
            1. this means anything after, in the code body, is skipped
        ii. keep in mind, if you have nested loops, a break statement will only exit the INNERMOST loop.

```
sum1=0
while(sum1<10):
  print("Sum is still less than 10")
  if(sum1==5):
    break;
    sum1=sum1*2 #unreachable code because of break stmt
  sum1+=1
```

iii.

```
Sum is still less than 10
Sum is still less than 10
Sum is still less than 10
Sum is still less than 10
Sum is still less than 10
Sum is still less than 10
```

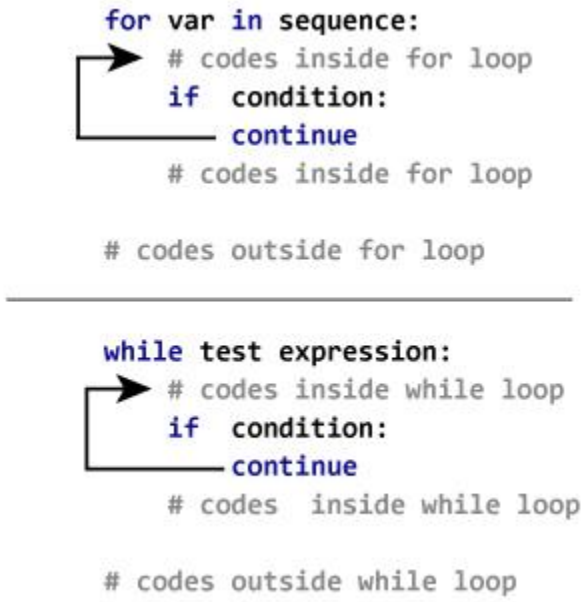
1. Executes only six times, because goes through the numbers 0-5, and when i=5, then the if stmt is true and the break stmt happens, breaking out of the while loop.

```
i=0
while(i <10):
  print("outer while loop running")
  while(i<5):
    print("inner while loop running")
    if(i==4):
      print("i is equal to 4, so will break")
      break
    i+=1
  i+=1
```

iv.

```
outer while loop running
inner while loop running
inner while loop running
inner while loop running
inner while loop running
inner while loop running
i is equal to 4, so will break
outer while loop running
outer while loop running
outer while loop running
outer while loop running
outer while loop running
```

18. Continue stmt:
    a. If continue is executed, then it will stop executing the body of the loop at that point, and return back to the top of the loop.
    b. Continue DOES NOT break like break does.



i.

```
for x in range(10,20):
  if(x%5==0):
    continue
  print(x)
```

ii.

```
11
12
13
14
16
17
18
19
```

Bibliography:

Ana Bell, Eric Grimson, and John Guttag. *6.0001 Introduction to Computer Science and Programming in Python.* Fall 2016. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu. License: Creative Commons BY-NC-SA.