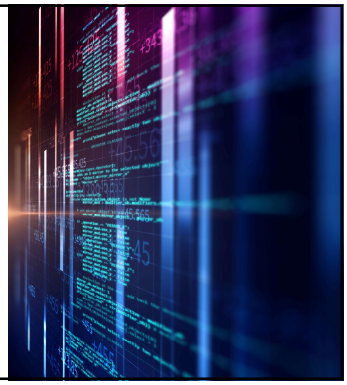
Loops

Brandon Krakowsky

Penn Engineering



1

Flow Control: Loops

2

Loops

- Used to repeat a process (block of statements) or perform an operation multiple times
- *for* loops
  - Run a piece of code *for* a given number of times
- *while* loops
  - Run a piece of code indefinitely *while* a condition is met

3

for Loops

- A *for* loop executes code a given number of times
- To do this, it iterates over a *list*

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for number in numbers:
    print(number)
```

- The *for* line indicates how many times the code will run
- *number* is a "dummy" variable that refers to the element in the list that we're passing through

Penn Engineering

4

for Loops

- We can iterate over the same *list* and find the numbers that are even

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers = []
for number in numbers:
    if (number % 2 == 0):
        even_numbers.append(number)

print(even_numbers)
```

- We initialized an empty list outside of the loop, then populated (appended) to the list as we iterated over the data

Penn Engineering

5

for Loops

- We can get a count of the even numbers by *incrementing* a count

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers = []
even_count = 0
for number in numbers:
    if (number % 2 == 0):
        even_numbers.append(number)
        even_count += 1

print(even_numbers)
print("There are", even_count, "numbers in the even list")
```

- Here's another way to get the count of even numbers

```
print(len(even_numbers))
```

Penn Engineering

6

for Loops - Exercise

- Write code that finds the minimum value of a list of numbers 5, 3, 8, -1, -2.2, 0
  - Don't use the built-in min() function
  - Instantiate a 'numbers' list variable containing the proper values (above)
  - Iterate over that list and find the min value
  - Print the minimum value in the format: "... is the min number"

```
numbers = [5, 3, 8, -1, -2.2, 0]
min_number = numbers[0]
for number in numbers:
if number < min_number:
  min_number = number

print(min_number, "is the min number")
```


7

for Loops

- You can iterate over *lists* of strings

```
planets = ['Sun', 'Mercury', 'Venus', 'Earth', 'Mars']
for planet in planets:
    if (planet == 'Sun'):
        print(planet, "is not a planet")
    else:
        print(planet, "is a planet")

    if (planet == 'Mercury'):
        print(planet, "is closest to the Sun")
```


8

for Loops

- You can also iterate over strings themselves!

```
month = "February"
print(month, "is spelled: ")
for x in month:
    print(x)
```


9

for Loops - Exercise

- Prompt the user for their first name
- Using a *for* loop
  - Print each letter of the name (on the same line)
  - Count each letter in the name
- Print the count of letters in the name

```
name = input("What is your first name?")
letter_count = 0
print(name, "is spelled:")
for x in name:
    print(x, end = ' ')
    letter_count += 1

print("There are", letter_count, "letters in", name)
```


10

for Loops Using *range*

- The *range* function generates a sequence (range) of numbers
  - This can be used, *like a list*, when you want to perform an action *n* number of times
- Format: range(*start*, *up_to*, *step*)
  - *start* and *step* are both optional
  - *up_to* means "up to but not including" the value
  - can only use integers! (no floats)
- Iterates over a sequence of 10 numbers, from 0 – 9

```
for x in range(10):
    print(x)
```

- This also iterates over a sequence of 10 numbers, from 0 – 9

```
for x in range(0, 10):
    print(x)
```


11

for Loops Using *range*

- Iterates over a sequence of 6 numbers, from 1 – 6

```
for x in range(1, 7):
   print(x)
```

- Iterates over a sequence of 5 numbers, from 0 – 28, skipping every 6 numbers

```
for x in range (0, 30, 7):
   print(x)
```

- Iterates over a sequence of 6 numbers, counting backwards from 5 – 0

```
for x in range (5, -1, -1):
   print(x)
```


12

for Loops Using *range*

- Here we find the numbers between 1 and 1200 that are odd

```
odd_numbers = []
for number in range(1, 1201):
    if (number % 2 != 0):
        odd_numbers.append(number)

print(odd_numbers)
```


13

while Loops

- A *while* loop repeatedly executes code based on a condition
  - Be careful -- if the condition is never met, your loop becomes an *infinite loop* and never stops
  - If this happens, your program could crash!
- This prints the value of a until it reaches 0

```
a = 5
while (a > 0):
    print("a is being decremented:", a)
    a -= 1
```

- Here's a program that multiplies x by 2 until an upper limit of 128, starting at 4

```
x = 4
while (x < 128):
    x = 2 * x
    print("x is now:", x)
```


14

while Loops – Getting User Input

- One use case for a *while* loop is a program that needs to continuously run and "wait" for something to happen, like specific user input
- This program runs until the user says 'hello'

```
inp = input('Hi! Please say hello.')
while inp != 'hello':
    inp = input('Please say hello.')

print('It\'s about time!')
```


15

while Loops - Exercise

- Write a program that uses a while loop to test user input of a secret password.
  - If the user inputs "secret", print "Welcome!" and exit the program
  - Otherwise, print "Sorry, the password you entered is incorrect. Please try again." and prompt the user again

```
password = ""
while password != "secret":
    password = input("Please enter the password:")
    if password == "secret":
      print("Welcome!")
    else:
      print("Sorry, the password you entered is incorrect.  Please try
again.")
```


16

Exit a Loop Using *break*

- *break* exits the entire loop immediately
- This prints 1- 4 only

```
x = 1
while x <= 10:
     if x == 5:
         break #this exits the entire while loop!

     print("x is now:", x)
     x += 1
```


17

Exit a Loop Using *continue*

- *continue* changes the flow of control and exits the current loop only
- This prints all of the odd numbers between 1 - 20, except those that are multiples of 3

```
for number in range(1, 21):
  if (number % 2 != 0):
    if (number % 3 == 0):
      continue #this exits the current iteration of the for loop only
    print(number)
```


18

Nested Loops

- A *nested loop* is a loop within a loop!
  - For every iteration of the *outer* loop, it runs the *inner* loop
- What does the following code print?

```
for i in range(1, 4):
    print('i:', i)
    for j in range(1, 3):
      print('\t', 'j:', j)
```


19

Nested Loops

- A *nested loop* is a loop within a loop!
  - For every iteration of the *outer* loop, it runs the *inner* loop
- What does the following code print?

```
for i in range(1, 4):
    print('i:', i) #for each number i
    for j in range(1, 3):
      #for each number j
      print('\t', 'j:', j)
```

```
i: 1
        j: 1
        j: 2
i: 2
        j: 1
        j: 2
i: 3
        j: 1
        j: 2
>>>
```


20

Nested Loops

- A *nested loop* is a loop within a loop!
  - For every iteration of the *outer* loop, it runs the *inner* loop
- What does the following code print?

```
for i in range(1, 4):
    print('i:', i) #for each number i
    for j in range(1, 3):
        #for each number j
        if (j <= 1):
            #continue to next iteration of current loop
            continue
        print('\t', 'j:', j)
```

```
i: 1
        j: 2
i: 2
        j: 2
i: 3
        j: 2
>>>
```


21

Nested Loops

- A *nested loop* is a loop within a loop!
  - For every iteration of the *outer* loop, it runs the *inner* loop
- What does the following code print?

```
for i in range(1, 4):
    print('i:', i) #for each number i
    for j in range(1, 3):
        #for each number j
        if (j <= 1):
            #break out of current loop only
            break
        print('\t', 'j:', j)
```

```
i: 1
i: 2
i: 3
>>>
```


22

Nested Loops – Multiplication Table Exercise

- Let's just make sure we know our multiplication tables

```
for i in range(1, 11): #for each number i (1 – 10)
    for j in range(1, 11): #iterate over each number j (1 – 10)
        print ("{} * {} = {}".format(i, j, i * j)) #multiply and print
```


23

Example Programs

24

Average Program

- Write a program that asks the user for numbers (ints). It computes the average of the numbers. Allows the user to enter -1 to quit the program.

```
num_list = []
i = 0
playing = True

#set up loop to repeatedly get user input of an int
while (playing == True):
    num = int(input("Enter num: "))
    if (num == -1):  #if the user inputs -1, this code will eventually
exit the loop
        playing = False

    num_list.append(num)
    i += 1
```


25

Average Program

- Write a program that asks the user for numbers (ints). It computes the average of the numbers. Allows the user to enter -1 to quit the program.

```
num_sum = 0
for num in num_list:
    num_sum += num

#calculate the average
num_avg = num_sum / i

print("avg:", num_avg)
```


26

Average Program

- What's wrong with this program?
- We're appending -1 (to exit the program) to our list of numbers in our loop

```
#set up loop to repeatedly get user input of an int
while (playing == True):
    num = int(input("Enter num: "))
    if (num == -1):  #if the user inputs -1, this code will eventually
exit the loop
        playing = False

    num_list.append(num)
    i += 1
```


27

Average Program

- What's wrong with this program?
- We're appending -1 (to exit the program) to our list of numbers in our loop
- Here's the fix!

```
#set up loop to repeatedly get user input of an int
while (playing == True):
    num = int(input("Enter num: "))
    if (num == -1):  #if the user inputs -1, this code will eventually
exit the loop
        playing = False
    else:
        num_list.append(num)
        i += 1
```


28

Word Reversal Program

- Write a program that reverses a word.

```
string = 'pasta'
rev = ''

#Iterates over a sequence, counting backwards from len(string) - 1 to 0
#with a step of -1
for j in range(len(string) - 1, -1, -1):
    rev += string[j]
print(rev)
```


29