


Repetition Structures (Ch. 4)

CSC110: INTRO TO COMPUTER PROGRAMMING WITH PYTHON

INSTRUCTOR: BILL BARRY



1

R o a d m a p		Iteration
		Midterm Exam (Th/F)
		Strings
		File I/O, Exceptions
		Lists
		List Algorithms
		Dictionaries & Sets
		Final Exam

2

COVID-19 Vaccine Attestations

North is a *fully vaccinated campus* per the Governor's higher-ed proclamation

All students must attest to their vaccination—or file for an exemption—by November 1st (Monday!). Students who don't do this **will not be able to register for Winter classes**, regardless of whether those classes are online, virtual, hybrid, or in-person

Only about half of our students have taken care of this so far. Let's fix that!

Instructions can be found here:

<https://www.seattlecolleges.edu/coming-campus/covid-19-vaccination-requirement>

Please take care of yours ASAP so you can register for Winter classes, available to you starting the same day, November 1st



3

This Week: Repetition (Loops)



Types of Loops,
Using for in Turtle

Practice with for

Counters &
Accumulators

Swapping variable
contents

while Loops

Pseudo-random
Numbers,
Sentinels, Nested
Loops

4

Loop Intro, Types of Loops, Using **for** in Turtle

5

Repetition/Iteration/Looping



Last week we learned a new control structure: Selection. Selection lets our code decide, based on data, whether to “flow” one way or a different way

But sometimes we need to make the same block of code execute several (or many) times. We’ve felt the pain of copying and pasting the same instructions in turtle again and again, for example

Now let’s learn how to make our code loop/repeat/iterate so that we can make our code more concise/clear, saying things like, “Execute the following code four times”

This is the last of the three control structures we discussed: sequence, selection, and repetition

6

Analogies: At the Dinner Table

FOR loop:

- "Eat five more bites of vegetables!"
- The child takes bites while counting 1, 2, 3, 4, 5
- This is a **definite** loop; we know (or can calculate) how many bites

WHILE loop:

- "Keep eating until your plate is clean!" ...or, closer to programming:
"While there is still food on your plate, keep taking bites!"
- This is an **indefinite** loop; we don't know exactly how many bites it will take
- The stopping point is specified by a condition; they need to keep going until that condition is met



7

Types of Repetition Structures

In Python there are two major programming constructs for repetition:

for loops (*definite* loops)

- *Collection-based loops*; stop when all collection elements have been processed
- You *don't* specify a loop condition or marking progress toward finishing; these manage themselves
- Note: these work a bit differently than **for** loops in other languages

while loops (*indefinite* loops)

- *Condition-based loops*; stop when a condition is met
- You specify the loop condition and ensure progress is moving in the right direction
 - It's easy to make infinite loops
- The condition test is at the top of the structure; must think a bit sometimes to make that work
- These are very similar to **while** loops in other languages

8

Anatomy of a **for** Loop

Python for loops often follow this pattern:

```
for variable in range(start, stop, [step]):
    loopStatements
```

—or—

```
for variable in range(stop):
    loopStatements
```

Read as "stop
just before"

What does range do? *generates an immutable sequence of numbers that can be **iterated***

What will this code print?

- ```
for number in range(5):
 print(number)
```
- ```
for number in range(3, 7):
    print(number)
```
- ```
for number in range(5, 15, 5):
 print(number)
```

Try these in IDLE:

```
len(range(1, 7, 2))
list(range(1, 7, 2))
```

We'll learn more about lists soon

9

## Anatomy of a **for** Loop, continued

Another typical pattern for Python for loops follows this syntax:

```
for variable in collection:
 loopStatements
```

Collections might include **lists** or tuples or even **strings**. What will this code print?

- ```
for number in [7, 3, 11, 5]:
    print(number)
```
- ```
greeting = "Hello"
for letter in greeting:
 print(letter)
```

10

# Pause & Play

---

LET'S WRITE SOME CODE USING FOR LOOPS

11

## Turtle Window Program: Old Code

Task: simplify our previous code by using loops

---

```
def drawSquare(size):
 forward(size)
 right(90)
 forward(size)
 right(90)
 forward(size)
 right(90)
 forward(size)
 right(90)
```

```
def drawWindow(squareSize):
 drawSquare(squareSize)
 left(90)
 drawSquare(squareSize)
 left(90)
 drawSquare(squareSize)
 left(90)
 drawSquare(squareSize)
 left(90)
```

12

## Turtle Window Program: Updated Code

---



```
def drawSquare(size):
 for side in range(0, 4):
 forward(size)
 right(90)
```

```
def drawWindow(squareSize):
 for side in range(0, 4):
 drawSquare(squareSize)
 left(90)
```

13

## Pause & Practice: using **for**

---

LET'S WRITE SOME CODE USING FOR LOOPS

14

## for Loop Practice

---

First let's remind ourselves...

- ...how concatenation works and use it to "build up" strings in a loop
- ...how Python's for loop can go through each letter in a string

A coded example including both: generate a password mask string ("hello" → "\*\*\*\*\*")

Then, starting with that concept/code, let's use Python **for** loops to accomplish these tasks

Write a function called **strReverse** that...

- ...accepts one parameter, a string
- ...returns the reverse of that string (e.g., turns "hello" into "olleh")

Write a function called **removeVowels** that...

- ...accepts one parameter, a string
- ...removes all the vowels from the string
- ...returns the result

15

## for Loop Practice: Bill's Solution #1

---



```
def strReverse(aString):
 revString = ""
 for ch in aString:
 revString = ch + revString
 return revString

def removeVowels(origString):
 origString = origString.upper()
 newString = ''
 for letter in origString:
 if letter != "A" and letter != "E" and letter != "I" \
 and letter != "O" and letter != "U":
 newString = newString + letter
 return newString
```

16





## for Loop Practice: Bill's Solution #2

---

```
def strReverse(aString):
 revString = ""
 for ch in aString:
 revString = ch + revString
 return revString

def removeVowels(origString):
 origString = origString.upper()
 newString = ''
 for letter in origString:
 if not (letter in ["A", "E", "I", "O", "U"]): # uses a list
 newString = newString + letter
 return newString
```

17

## Counters and Accumulators

---

18

## Intro to Counts

---

What does this line of code do, assuming you already have a variable called myCount?

```
myCount += 1 # same as myCount = myCount + 1
```

What if we put the same statement inside a loop?

```
myCount = 0 # why is this necessary?
for item in _____:
 # ...maybe other stuff here...
 myCount += 1
```

What will myCount contain after the loop ends?

19

## Counts and Totals

---

### Count

- Before the loop, create a count variable and set it to zero
- Inside the loop, **add one** to the count variable
- Outside the loop, return or display the count

### Total (aka Accumulation)

- Before the loop, create a total variable and set it to zero
- Inside the loop, **add the value** (whatever you're totaling) to the total variable
- Outside the loop, return or display the total

Example (what will this print?):

```
total = 0
for aNum in range(1, 5):
 total += aNum
print(total)
```

20

# Pause & Practice

---

LET'S WRITE SOME CODE USING ACCUMULATORS

21

## for Loop and Accumulator Practice

Use Python for loops to accomplish these tasks

Write a function called **addNums** that...

- ...accepts two parameters, a starting and ending number (integers)
- ...adds up all the numbers in the range startNum...endNum (inclusive), returning the sum

22



## addNums: Bill's Solution

---

```
def addNums(startNum, endNum):
 total = 0
 for aNum in range(startNum, endNum + 1):
 total += aNum
 return total
```

Note: a pro Python programmer wouldn't write a loop for this; Python has a built-in way. With a single keyword, you can sum a whole range! Not all languages offer this ease

```
sum(range(1, 11)) # 55
```

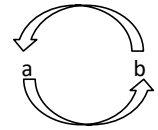
23

## Swapping Variable Contents

---

24

## Swapping Variable Contents: Analogy



Imagine two white boards (A & B), each with a word written on them. You need to swap them, but you aren't allowed to memorize either of the words to do that. You can copy from one board to another, though

Giraffe    Lion

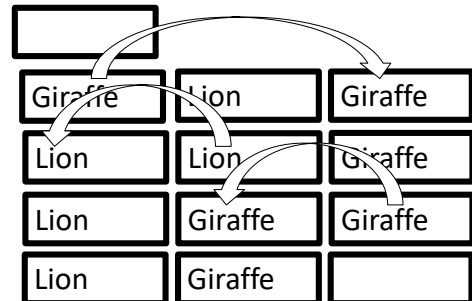
What you need is a temporary white board (Temp)

First, copy A's text to Temp

Now, copy B's text to A

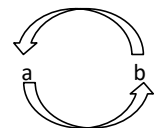
Finally, copy Temp's text to B

Now you can put away the temporary one



25

## Swapping Variable Contents: Code



Let's say you have two variables, student1 and student2

```
student1 = "Parker"
student2 = "Ash"
```

You want to swap their values. Here's what you might try:

```
student1 = student2 // student1 gets "Ash"
student2 = student1 // student2 gets "Ash"
```

Oops, you lost a student!

You need a temporary variable for swapping:

```
temp = student1 // temp gets "Parker"
student1 = student2 // student1 gets "Ash"
student2 = temp // student2 gets "Parker"
```

26

# Python **while** Loops

---

LET'S LOOK AT HOW PYTHON'S WHILE LOOPS WORK

27

## **while** Loops

---

Earlier this week we talked about two types of loops:

- **Definite** loops, where we know or can count the number of iterations required
- **Indefinite** loops, where we can't count the iterations required, but are instead focused on a goal

Python's **for** loop is perfect for **definite** scenarios; we'll always use that if we can

For **indefinite** scenarios, Python offers the **while** loop

Some examples of indefinite scenarios where we'll use **while**:

- Gaming, where the winner is determined by score (regardless of how many turns/rounds)
- Inputting employee data for payroll, processing each employee until there are no more
- Validating input, where we keep asking a question until they give an in-range answer
- Writing out mathematical sequences where we have an end goal but can't count (e.g., Fibonacci)

**while** is a common keyword across languages, too (JavaScript, Java, C)

28

# Python Loops

## FOR

You have **one** job:

Provide something Python can loop through ("iterate over", aka an *iterable*)

The worst that can happen is the loop doesn't run at all

It's impossible to make an infinite loop

For these reasons, always use this type of loop when possible

## WHILE

You have **three** jobs:

1. Before the loop, set up conditions for entry into the loop (or not)
2. Set up a correct loop condition
3. Inside the loop, make progress toward the exit (making the condition false)

It's possible to create loops that run *infinitely*

- Use <Ctrl+C> if you get into an infinite loop

29

# Anatomy of a **while** Loop

Python while loops follow this syntax:

```
while condition:
 loopStatement(s)
```

Don't forget to *make progress toward making the condition False* or you'll loop infinitely

What does this code display?

```
num = 5
total = 0
while num < 10:
 total += num
print(total)
```

```
stars = ""
while len(stars) < 10:
 stars += " "
print(stars)
```

Can you identify the three jobs in the code samples?

Both examples would be best suited to **for**, but we're just learning the syntax...

30

# Pause & Play

---

LET'S WRITE SOME CODE USING WHILE LOOPS

31

## Python while Loop Practice

Fibonacci sequences are generated by working with pairs of numbers, starting with 0 and 1. You generate the next number by adding them together (1, in this case). You then move forward working with the last pair of numbers, 1 and 1, which add together to make 2. Then continue

Write a function called **displayFibonacci** that...

- ...accepts one parameter, the maximum number you wish to generate
- ...generates and displays a Fibonacci sequence through the specified number

For example, if called with 8 as the argument, it would display this:

- 0 1 1 2 3 5 8

...and, if called with 15:

- 0 1 1 2 3 5 8 13

32





## displayFibonacci: Bill's Solution

```
def displayFibonacci(limit):
 num1 = 0
 print(num1, end = ' ')

 num2 = 1 ①
 while num2 <= limit: ②
 print(num2, end = ' ')
 # set up next in sequence
 next = num1 + num2
 num1 = num2
 num2 = next ③
 print()
```

33

## Implications of “Test at the Top”

Most languages offer two types of indefinite loops:

- Test at the **Top**—perfect for cases where you may not need to execute the loop at all (0+)
- Test at the **Bottom**—perfect for cases where must execute the loop at least once (1+)

Python offers just the one type: test at the Top

Consider these two scenarios:

- Do you want habanero pepper sauce on your food? If so, add spoonful and ask again
- We give you a scoop of ice cream, then ask if you'd like another

How do we code them?

On “must do one time” loops we must ensure we get into the loop the first time, or do one execution *outside the loop*, first



34

## Test at the Top: Pseudocode

---

### HABANERO SAUCE

```
ask if they want a spoonful
while (answer = yes)
 give a spoonful
 ask if they want another
```

### ICE CREAM: OKAY, BUT REPETITIVE

```
give one scoop
ask if they want another
while (answer = yes)
 give one scoop
 ask if they want another
```

35

## Test at the Top: Pseudocode

---

### HABANERO SAUCE

```
ask if they want a spoonful
while (answer = yes)
 give a spoonful
 ask if they want another
```

### ICE CREAM: BETTER

```
answer = yes
while (answer = yes)
 give one scoop
 ask if they want another
```

36

# Pseudo-random Numbers, Sentinels, Nested Loops

---

37

## Pseudo-random Numbers

---

For games and simulations, it's often useful to generate random numbers

These aren't truly random; the computer uses a *seed* (usually the computer's internal clock) to start the sequence. This generates a floating-point number in the range 0.0 to just under 1.0:

```
import random
print random.random()
```

How can we generate an integer in the range 1 to 10 (inclusive)? There's a function for that!

```
import random
randNum = random.randrange(1, 10 + 1)
print(randNum)
```

If you want to generate the same numbers each time, use and remember a "seed" value:  
`random.seed(32)`

38

## Python **while** Loop Practice

---

Write a function called rollDie that generates a random roll of a die (1 to 6)

Write a function called rollDieUntil that accepts one parameter (an integer) that indicates what roll of the die the caller is looking for. It keeps rolling the die and showing the results until the target roll is seen

For example, a call to rollDieUntil(5) might show this result:

Roll: 1

Roll: 4

Roll: 3

Roll: 4

Roll: 2

Roll: 5

39

## rollDie: Bill's Solution

---



```
import random

def rollDie():
 dieRoll = random.randrange(1, 7)
 return dieRoll

def rollDieUntil(targetRoll):
 print("Your roll target is", targetRoll)
 roll = -99 ①
 while roll != targetRoll: ②
 roll = rollDie() ③
 print("Roll:", roll)
 print("You rolled the target number!")

rollDieUntil(5)
```

40

## Sentinels

---

A **sentinel** value is value that signals the process that something special has happened

Examples:

- Loop lets you enter employees and work hours until you provide an employee # of '0000'
- The last record in the file to be read has string 'EOF'
- A menu makes you choose until you select the exit option

Let's tackle the first example. Let's write code that...

- ...lets us enter an employee number
- ...lets us enter the number of hours worked that week
- ...exits when we enter an employee number of '0000'
- ...displays the total hours worked, for all employees

41

## Employee Payroll with Sentinel

---

```
totalEmpHours = 0
empNum = input('Enter employee # or 0000 to exit: ') ①
while empNum != '0000': ②
 empHours = float(input('Enter employee hours: '))
 totalEmpHours += empHours
 empNum = input('Enter employee # or 0000 to exit: ') ③
print('Total employee hours --> ', totalEmpHours)
```

42

## Input Validation Loops

It's often desirable to keep the user "trapped" until they do what you ask

Example: menu system requires a valid choice; we want to lock them in until they comply

```
Employee Records Menu
1. Search for employee
2. Add employee
3. Delete employee
0. Exit
Enter your choice: _
```

Let's do this one together

43

## displayMenu: Code



```
def doMenu():
 print('Employee Records Menu')
 print()
 print('1. Search for employee')
 print('2. Add employee')
 print('3. Delete employee')
 print('0. Exit')
 print()
 menuChoice = -99 #seed value to get us into the loop ①
 while menuChoice < 0 or menuChoice > 3: ②
 menuChoice = int(input('Select option: ')) ③
 return menuChoice
```

44

## Nested Loops

It is sometimes useful to put a loop inside another loop; this is called “nesting”

- There’s an “outside loop” that runs more slowly, and an “inside loop” that runs more quickly

Analogy #1: you want to run 2 miles, but the track is only  $\frac{1}{4}$  of a mile.

- In your mind, you might say, “Starting mile 1: lap 1, lap 2, lap 3, lap 4. Now starting mile 2: lap 1, lap 2, lap 3, lap 4” You have two counters in your head: one for miles (ticks slowly), a second one for laps (ticks faster)

Analogy #2: car odometer; tenths click by quickly, ones a bit more slowly, tens slower yet, etc.

- Video (thanks, Beth!): <https://canvas.northseattle.edu/files/160885599>

Example: print a multiplication table for your child to use as a study aid

Let’s do this one together

45

## displayMultTable: Code

```
def displayMultTable(maxNum):
 for row in range(1, maxNum+1):
 for col in range (1, maxNum+1):
 print(format(row * col, '5d'), end='')
 print()
```

|   |    |    |    |    |
|---|----|----|----|----|
| 1 | 2  | 3  | 4  | 5  |
| 2 | 4  | 6  | 8  | 10 |
| 3 | 6  | 9  | 12 | 15 |
| 4 | 8  | 12 | 16 | 20 |
| 5 | 10 | 15 | 20 | 25 |

Some text or online examples will use “i” and “j” for variables; this is a pet peeve of mine; there is always context! Variables deserve descriptive names, which will also keep you from having to “decode” what the letters mean

46

The End

---

