Lab 4 Feedback

- We need some work on functions
- Follow examples and instructions

Refactoring: Displaying Fibonacci Sequence

What part of this code needs to go into the function?
What is the input to the function?
What is the output from the function?

```
print("Displays the first 20 Fib nums…")

prevNum2 = 0
prevNum = 1

print(prevNum2)
print(prevNum)

for i in range(18) :
    fibNum = prevNum + prevNum2
    print(fibNum)
    prevNum2 = prevNum
    prevNum = fibNum
```

Refactoring: Displaying Fibonacci Sequence

What part of this code needs to go into the function?
What is the input to the function?
What is the output from the function?

Unintended side effect

```
print("Displays the first 20 Fib nums…")

prevNum2 = 0
prevNum = 1

print(prevNum2)
print(prevNum)

for i in range(18) :
    fibNum = prevNum + prevNum2
    print(fibNum)
    prevNum2 = prevNum
    prevNum = fibNum
```

Code that displays the Fibonacci sequence

Doc String for Fibonacci Sequence Function

- How should we describe this function?
  - What is a good precondition for the function?
    - What info does a good precondition include?

```
def generateFibonacciNumber(numInSequence):
    """

    """
```

Doc String for Fibonacci Sequence Function

- How should we describe this function?
  - What is a good precondition for the function?
    - What info does a good precondition include?

```
def generateFibonacciNumber(numInSequence):
    """
    Pre: numInSequence must be an integer greater than 1
    Post: returns the numInSequence value
          in the Fibonacci sequence
    """
```

Does not mention user input – does not require user input.

Doc String for Fibonacci Sequence Function

- How should we describe this function?
  - What is a good precondition for the function?
    - What info does a good precondition include?

```
def generateFibonacciNumber(numInSequence):
    """
    Pre: numInSequence must be an integer greater than 1
    Post: returns the numInSequence value
          in the Fibonacci sequence
    """
```

Does not mention user input – does not require user input.

```
for x in range( 2, 10, 2):
    print( generateFibonacciNumber(x) )
```

Molecular Weight

- Given a non-negative integer of hydrogen, oxygen, carbon atoms, return the molecular weight

```
def calcMolecularWeight( hAtoms, oAtoms, mAtoms ):
    ... # calculation ...

    return weight
```

Rounding should not be done in here
→ Reduces the reusability of the function

Molecular Weight

- Given a non-negative integer of hydrogen, oxygen, carbon atoms, return the molecular weight

```
def main():
    # get user input …
    weight = calcMolecularWeight(...)
    print("The weight is", round(weight, 6))
```

Would still only round to 3 places
if rounding performed in function

Review

- How can we make our code make [good] decisions?

Grade – separation of concerns

- If with the ands compared to the if/else

More Complex Conditions

- Boolean
  - Two logical values: True and False
- Combine conditions with Boolean operators
  - **and** – True only if **both** operands are True
  - **or** – True if **at least one** operand is True
  - **not** – True if the operand is not True
- English examples
  - If it is raining **and** it is cold
  - If it is Saturday **or** it is Sunday
  - If the shirt is on sale **or** the shirt is purple

What is the output?

Focus: how operations work
Not good variable names

```
x = 2
y = 3
z = 4

b = x==2
c = not b
d = (y<4) and (z<3)
print("d=",d)
d = (y<4) or (z<3)
print("d=",d)

d = not d
print(b, c, d)
```

Because of precedence, we don't *need* parentheses

Truth Tables

operands

A	B	A and B	A or B	not A	not B	not A and B	A or not B
T	T						
T	F						
F	T						
F	F						

Truth Tables

operands

A	B	A and B	A or B	not A	notB	not A and B	A or not B
T	T	T	T				
T	F	F	T				
F	T	F	T				
F	F	F	F				

Truth Tables

operands

A	B	A and B	A or B	not A	notB	not A and B	A or not B
T	T	T	T	F	F		
T	F	F	T	F	T		
F	T	F	T	T	F		
F	F	F	F	T	T		

Truth Tables

operands

A	B	A and B	A or B	not A	notB	not A and B	A or not B
T	T	T	T	F	F	F	T
T	F	F	T	F	T	F	T
F	T	F	T	T	F	T	F
F	F	F	F	T	T	F	T

Practice: Numeric Grade Input Range

- Enforce that user must input a numeric grade between 0 and 100
  - In Python, we can't (always) write a condition like 0 <= num_grade <= 100, so we need to break it into two conditions
- Write an appropriate condition for this check on the numeric grade
  - Using **and**
  - Using **or**

Focus on the ***condition***
Then, we'll block out the code

Practice: Numeric Grade Input Range

- Enforce that user must input a numeric grade between 0 and 100
  - Using **and**

```
if num_grade >= 0 and num_grade <= 100:
    computation
else:
    print error message
```

  - Using **or**

```
if num_grade < 0 or num_grade > 100:
    print error message
else:
    computation
```

Lab 5 Overview

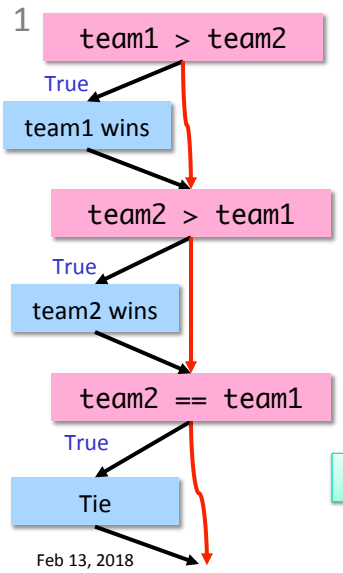
- "only" two non-exam class periods since last lab, so…
- Focus on conditionals
- More building blocks to draw from
  - Break problem into smaller pieces
  - Think, write your algorithm outline, write a few lines of code, then try them out.
- Table functions for a week

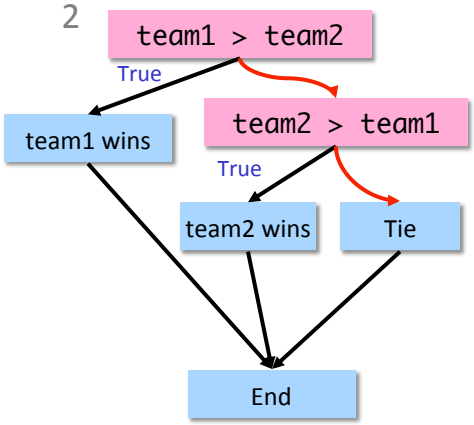
Common Issue: Inefficiency

```
if team1Score > team2Score:
    print("Team 1 wins!")
else:
    if team2Score < team1Score:
        print("Team 2 wins!")
    else:
        if team1Score == team2Score:
            print("They tied! We're going to overtime!")
```

Extra if statement, not necessary
Know when hit second else that the only possibility is a tie

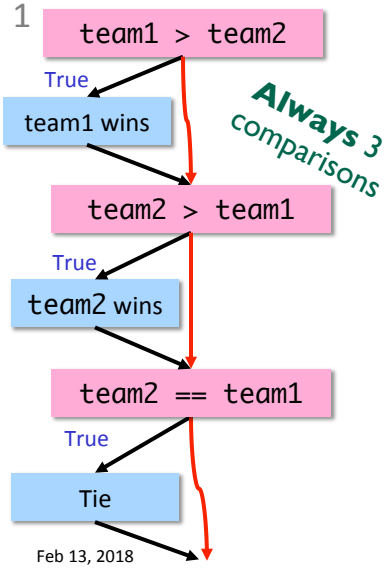
Problem 1, 2 Efficiency

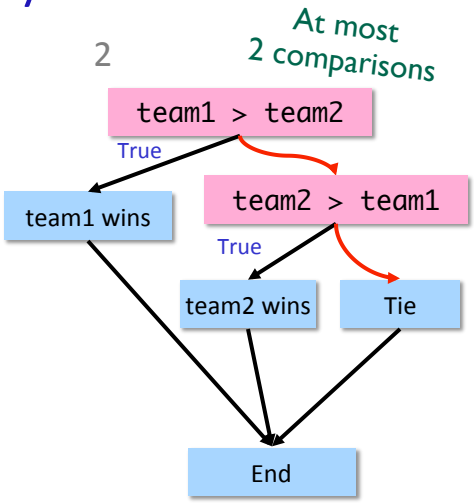




- How many conditions evaluated?

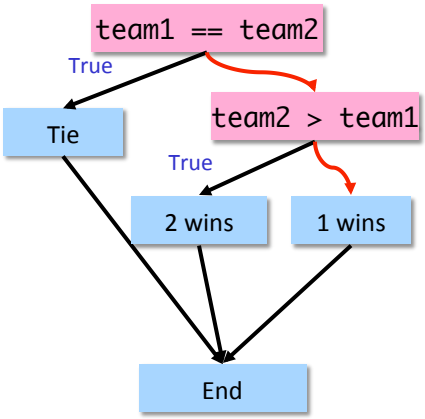
Problem 1, 2 Efficiency

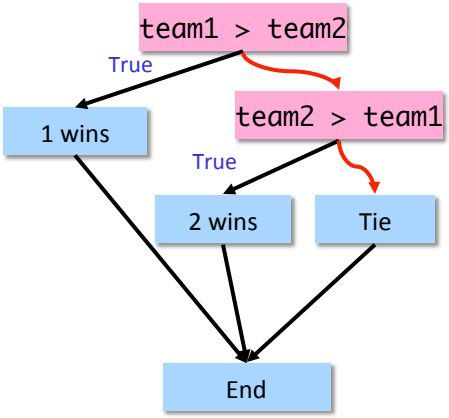




Problem 2 (& 3) Efficiency

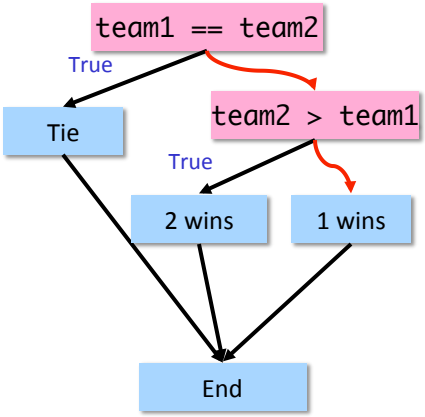
Which tends to be more efficient?
How many conditions to evaluate?



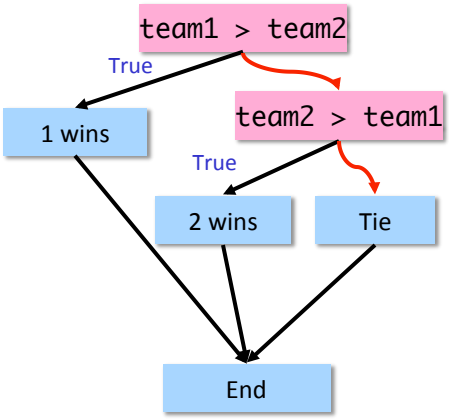


Problem 2 (& 3) Efficiency

Equality is a rare condition; on average, will always need to check second condition.



More common case. May only need to check one condition.



Adding to Development Process

- Last development step:
  - Assess your program again after it works
  - Is it efficient? Is it readable? Can I simplify?

Lab 4 – Greatest Hits: Less-Complicated Approaches for Customized Display

- Correct but more complicated solution to handling customized display

Other, similar examples in submissions

```
if albums == 1 and extraTracks == 0:
    print("Your album requires", albums, "cd")
elif albums == 1 and extraTracks > 0:
    print("Your album requires", albums, "cd")
    print(extraTracks, "tracks will have to wait for
                        the next Greatest Hits album")
elif albums > 1 and extraTracks > 0:
    print("Your album requires", albums, "cds")
    print(extraTracks, "tracks will have to wait for
                        the next Greatest Hits album")
elif albums > 1 and extraTracks == 0:
    print("Your album requires", albums, "cds")
```

Lab 4 – Greatest Hits: Less-Complicated Approaches for Customized Display

- Less complicated solution
  - Simpler logic, conditions
  - Less duplicated code

```
if albums == 1:
    print("Your album requires", albums, "CD.")
else:
    print("Your album requires", albums, "CDs")

if extraTracks > 1:
    print(extraTracks, "tracks will have to wait for
                    the next Greatest Hits album")
elif extraTracks==1:
    print(extraTracks, "track will have to wait for
                    the next Greatest Hits album")
```


REVIEW: STRINGS

Review

- How can we combine strings?
- How can we find out how long a string is?
- How can you tell if one string is contained in another string?
- How can we find out the character at a certain position?
- How can we iterate through a string?
- How do you call a method on a string?

String Operations

Operand	Syntax	Meaning
+	`str1 + str2`	Concatenate two strings into one string
*	`str * num`	Concatenate string `num` times

- Examples:
  - `"I feel " + "sleepy"`
    - Evaluates to `"I feel sleepy"`
  - `"Oops! " * 3`
    - Evaluates to `"Oops! Oops! Oops! "`

String Comparisons

- Same operations as with numbers:
  - ==, !=
  - <, <=
  - >, >=

  } Alphabetical comparison

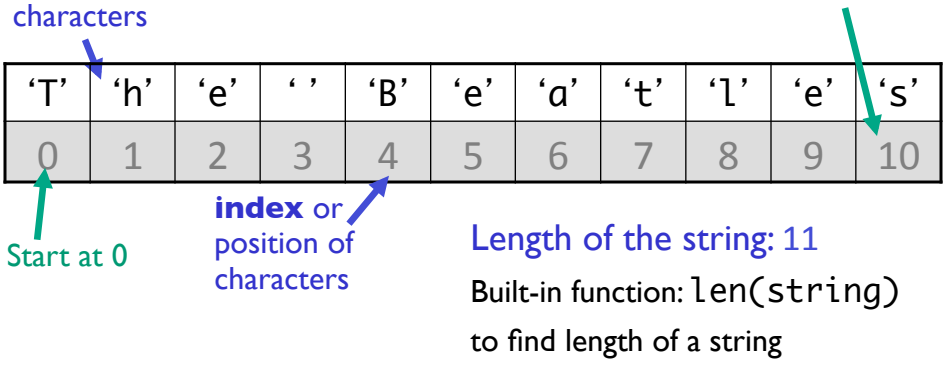
- Use in conditions in `if` statements

```
if userpick == pick4num:
    print("We have a winner!")
else:
    print("You lose.")
```


Strings

- A *sequence* of characters
  - Example:

band = "The Beatles"



Length of the string: 11

Built-in function: `len(string)` to find length of a string

Summary: Iterating Through a String

- For each character in the string

string of length 1

```
for char in mystring:
  print(char)
```

Determines loop's behavior

- For each position in the string

An integer

```
for pos in range(len(mystring)):
  print(mystring[pos])
```

Index into the string

`str` Methods

- Example method: **`find(substring)`**
  - Finds the index where substring is in string
  - Returns -1 if substring isn't found

- To call a method:
  - `<str_obj>.methodname([arguments])`
  - Example: `filename.find(".py")`

Executed on this string

Common `str` Methods

Method	Operation
`center(width)`	Returns a copy of string centered within the given number of columns
`count(sub[, start [, end]])`	Return # of non-overlapping occurrences of substring `sub` in the string.
`endswith(sub), startswith(sub)`	Return `True` iff string ends with/starts with `sub`
`find(sub[, start [, end]])`	Return first index where substring `sub` is found
`isalpha(), isdigit(), isspace()`	Returns `True` iff string contains letters/ digits/whitespace only
`lower(), upper()`	Return a copy of string converted to lowercase/lowercase

 `string_methods.py`

Common `str` Methods

Method	Operation
`replace(old, new[, count])`	Returns a copy of string with all occurrences of substring `old` replaced by substring **`new`**. If `count` given, only replaces first `count` instances.
`split([sep])`	Return a list of the words in the string, using `sep` as the delimiter string. If `sep` is not specified or is None, any whitespace string is a separator.
`strip()`	Return a copy of the string with the leading and trailing whitespace removed
`join(<sequence>)`	Return a string which is the concatenation of the strings in the sequence with the string this is called on as the separator
`swapcase()`	Return a copy of the string with uppercase characters converted to lowercase and vice versa.

Using the APIs

- Given a problem, break down the problem
  - Can any of the parts of the problem be solved using a method in the API?

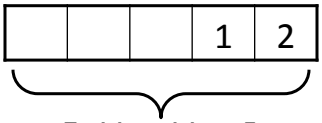
Escape Sequences

- Escape character: \
- Escape sequences
  - newline character (carriage return) → `\n`
  - tab → `\t`
  - quote → `\"` or `\'`
  - backslash → `\\`
- Example:
  - `print("To print a \\, you must use \"\\\\\"")`
    - What does this display?

FORMATTING STRINGS

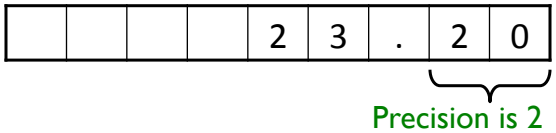
Example Format Specifiers

```
"{:5d}".format(12)
```
→ " 12"



Field width is 5

```
"{:9.2f}".format(23.1999)
```
→ " 23.20"



Precision is 2

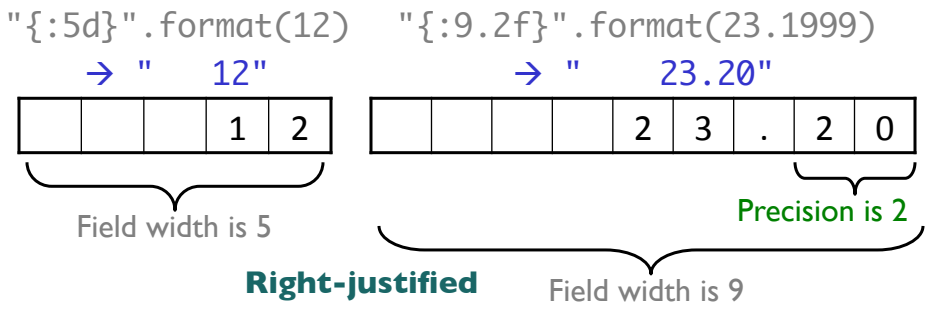
Field width is 9

Right-justified

- What if precision is bigger than the decimal places?
- What if field width is smaller than the length of the value?

Any guesses? Try out in interpreter.

Example Format Specifiers



- What if precision is bigger than the decimal places?
  - Fills decimal with 0s
- What if field width is smaller than the length of the value?
  - String contains entire value

Formatting Practice

- `x = 10`
- `y = 3.5`
- `z = "apple"`
- `"{:6d}".format(x)`
- `"{:6.2f}".format(x)`
- `"{:06.2f}".format(y)`
- `"{:6.2f}".format(y)`
- `"{:^10s}".format(z)`
- `"{:5d} {:<7.3f}".format(x,y)`

Example: Printing Out Tables

- A table of temperature conversions

```
Temp F      Temp C      Temp K
------      ------      ------
-459.7      -273.1         0.0
   0.0       -17.8       255.2
  32.0         0.0       273.1
```

- If we want to print data in rows, what is the template for what a row looks like?
  - How do we make the column labels line up?

Course Midterm Grades

- For those of you who get midterm grades, I will calculate your grade based on
  - 50% midterm exam
  - 40% labs (through lab4 at least)
  - 5% broader issues
  - 5% participation

Lab 5

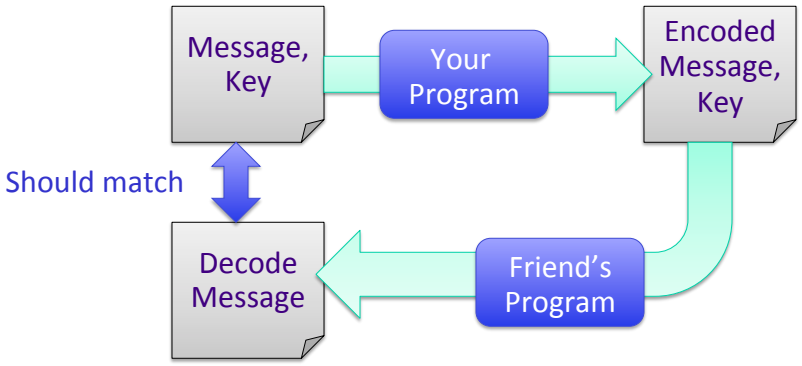
- Basic and advanced string problems

Review

- How do we get fine-grained control over how to format your output?
- How do you call a method on a string?
- How do you convert from a character to its decimal ASCII representation?
- How do you convert from a decimal ASCII representation to the character?

Caesar Cipher

- Write an encoding/decoding program
  - Encode a message
  - Give to a friend to decode



Lab 5 Overview

- More String Problems
- ASCII manipulation

Problem with Duplicate Code

```
#Print data for 100m
meters = 100
kilometers = .001*meters
yards = 1.094*meters
miles = .0006215*meters
print("%5d %12.3f %13.1f %10.3f" %
         (meters,kilometers,yards,miles))

#Print data for 200m
meters = 200
kilometers = .001*meters
yards = 1.094*meters
miles = .0006215*meters
print("%5d %12.3f %13.1f %10.3f" %
         (meters,kilometers,yards,miles))

…
```

Difficult to maintain:
- What if need to change the formatting?
- Change the conversion amount?

Solutions:
- Constants
- For loop

BMI problem

- Compute BMI as a *float*

Why is that important?

- Check valid height, weight *before* computing BMI
  - Don't waste time computing if they are invalid
- Most elegant if/else condition:

```
if bmi < 19:
   print "below"
elif bmi > 25:
   print "above"
else:
   print "in range"
```

As opposed to

```
if bmi >= 19 and bmi <= 25:
   print "in range"
elif bmi > 25:
   print "above"
else:
   print "below"
```

Both are correct

Partial Student Solution

```
TOO_HEAVY = 400
TOO_LIGHT = 60
TOO_TALL = 84 # 7ft
TOO_SHORT = 36 # 3 ft

LOW_BMI = 19
HIGH_BMI = 25

print "Do you know your BMI? You should!\n"

height = input ("How tall are you in inches? ")
# Check for unreasonable input, exit if unreasonable
if height <= TOO_SHORT:
    print "You can't be that short!"
    sys.exit(1)
elif height >= TOO_TALL:
    print "You can't be that tall!"
    sys.exit(1)
```

Well-named constants
Easy to change values
Also explained values (e.g., 7ft)

Good error messages for unreasonable cases

Checks before user enters weight; less work for user

Exam 1 Results

	A	B	C	Total
Average	77	75	80	83
Median	76	75	84	86

- Had 104 points but out of 100, plus 6 bonus points
- Common mistakes
  - Budgeting time
    - More than I asked for
  - Not answering part of the question
  - Tracing through `if` problem, fixing code
    - use control flow diagrams
  - `while` loop → `for` loop
  - <= instead of ≤

Tip Chart

You didn't have to worry about column widths

```
print "Meal Cost   15% tip   20% tip"

for mealCost in xrange(10, 105, 5):
    tip15 = mealCost * .15
    tip20 = mealCost * .20
    print "%9d %9.2f %9.2f" % (mealCost, tip15, tip20)
```


Animal Adoption

```
num = input("Enter the number of cats you want to adopt: ")
fixed = input("Enter 1 if the cats are neutered or 0 if they
aren't: ")

adoption_fee=65

if num > 1:
    adoption_fee -= 5*num

if fixed == 1:
    adoption_fee += 20

total = adoption_fee * num

# Solution does not require the decimal formatting
print "The adoption fee per animal is $%.2f" % adoption_fee
print "The total adoption fee is $%.2f" % total
```

Note how clean/simple the solution is

Grading

- (38%) Programming projects
- (30%) Two hourly exams
- (20%) A comprehensive final exam
- (7%) Writeups and discussions of CS-related issues
- (5%) Participation and attendance

String Review

- How do we call methods on a string?
  - How can we find out the methods that are available?
- How can we tell if some string is part of some other string?

Lab 4 Feedback

- Problem 5, if number is divisible by 6
  - Prefer `while num % 6 != 0 :`
    to `while not (num % 6 == 0) :`
- Separate blocks of code with *spaces* and *comments*
  - More important as we write larger programs
  - Especially with graphics programs
- Craps: not printing values of rolls

Lab 4 Feedback

```
number = 1
while number % 6 != 0:
   number=input("Enter a number …: ")
   if number % 6 != 0:
        print number, "is not divisible by 6. ... "

print number, "is divisible by 6."
```

Checking same condition

Change order of statements:

```
number=input("Enter a number …: ")
while number % 6 != 0:
   print number, "is not divisible by 6. ... "
   number=input('Enter a number …: ')

print number, "is divisible by 6."
```

Input Restrictions

```
user_num=int(input("Please input a number."))
```


Simplifying Code

```
word=str(input("Enter a word: "))

#create word in clunky pig latin
#slice string so that first letter is moved to the end
first_letter=word[0:1]
word_slice=word[1:]

#add 'ay' to the end of the word
pig_latin=word_slice+first_letter+"ay"

print("In clunky pig latin, that word is", pig_latin)
```

How can we simplify this code?

Consider The Following Solution

```
string1 = input("Enter the first word: ")
string2 = input("Enter the second word: ")
string3 = input("Enter the third word: ")

if string1 <= string2:
    if string1 <= string3:
        first=string1
    else:
        first = string3
elif string2 <= string3:
    first=string2
else:
    first=string3

print("The alphabetically first word is " +first)
```

Is the above solution correct?
How efficient is the solution?
Any good characteristics you notice?

Drawing a Box Alternatives

- For loop

```
print("."*width)
for rowNum in range(height-2):
    print("." + " "*(width-2) + ".")
print("."*width)
```

- str operations

```
print("."*width)
line = "." + " "*(width-2) + ".\n"
print(line*(height-2), end='')
print("."*width)
```

Error Handling

```
WIDTH_INPUT = "Enter a width (" +str(WMIN)+"-"+str(WMAX) + "): "
HEIGHT_INPUT ="Enter a height ("+str(HMIN)+"-"+str(HMAX) + "): "

width = int(input(WIDTH_INPUT))
height = int(input(HEIGHT_INPUT))

error = False
errorMessage = "\nError: \n"

if width < WMIN or width > WMAX:
    error = True
    errorMessage += "\tWidth (" +str(width) + ") is not within
                    range (" + str(WMIN) + "-" + str(WMAX) + ")\n"

if height < HMIN or height > HMAX:
    error = True
    errorMessage += "\tHeight (" +str(height) + ") is not within
                   range ("+ str(HMIN) + "-" + str(HMAX) + ")\n"

if error:
    print(errorMessage)
    sys.exit(1)
```

Simplifying Code

```
if (width >= 2 and width <= 80) and (height>=2 and height<=80):
    print('.' * width)
    if height > 2:
        for num in range(height-2):
            print('.' + (' '*(width-2)) + '.')
    print('.' * width)
else:
    # error message and exit …
```

How can we simplify this code?