Compsci 101
DeMorgan’s Law, Short circuiting,
Global, Tuples

Susan Rodger
February 17, 2022

L is for …

• Loops
  • While, For, Nested – Iteration!

• Library
  • Where we find APIs and Implementations

• Logic
  • Boolean expressions in if statements, loops

• Linux
  • The OS that runs the world?

Keith Kirkland

• BS ME, BFA Accessories Design, MID Industrial and Product Design
• Co-founder of WearWorks
• Wayband – wearable haptic navigation device for blind
• Device guided blind marathon runner in NYC marathon

“We design products that shift people’s lives in a meaningful way”

Announcements

• APT-3 due tonight
• Assign 3 due Tuesday, March 1
• Lab 6 on Friday, do prelab

• Exam 2 – Tuesday, Feb 22
  • in person during lecture time
• APT Quiz 1 – Feb 24-27
Exam 2 – in person – Tues, Feb 22

• Exam is in class on paper – 10:15am
  • Need pen or pencil
• See study materials under 2/22 date
  • Exam 2 Reference sheet - part of exam
• Covers
  • topics /reading through Tues. Feb 15
  • APTs through APT3
  • Labs through Lab 5, Lab 6 (Part 1 – list comprehensions)
  • Assignments through Assignment 2
  • Concepts from Assign2, No turtles

Exam 2 – How to Study

• Practice writing code on paper!
• Rewrite an APT
• Try to write code from lecture from scratch
• Try to write code from lab from scratch
• Practice from old exams
• Put up old Sakai quizzes, but better to practice writing code
• Look at Exam 2 reference sheet when writing code!

Exam 2

• Exam 2 is your own work!
• No looking at other people’s exam
• You cannot use any notes, books, computing devices, calculators, or any extra paper
• Bring only a pen or pencil
• The exam has extra white space and has the Exam 2 reference sheet as part of the exam.

• Do not discuss any problems on the exam with others until it is handed back

APT Quiz 1 Feb 24-27

• Opens 2/24 11:30am
• Closes at 11pm 2/27 – must finish all by this time
• There are two parts based on APTs 1-3
  • Each part has two APT problems
  • Each part is 1.5 hours – more if you get accommodations
  • Each part starts in Sakai under tests and quizzes
  • Sakai is a starting point with countdown timer that sends you to a new apt page just for each part
  • Could do each part on different day or same days
• Will put up problems today from an old APT Quiz so you can practice (not for credit) – on APT Page
APT Quiz 1

- Is your own work!
  - No collaboration with others!
  - Use your notes, lecture notes, your code, textbook
  - DO NOT search for answers!
  - Do not talk to others about the quiz until grades are posted
- Post private questions on Ed Discussion
  - We are not on between 10pm and 8am!
  - We are not on all the time
  - Will try to answer questions between 8am – 10pm
- See 101 APT page for tips on debugging APTs

PFTD

- Tuples
- Global
- DeMorgan’s Law
- Short Circuiting
- APT

Tuple: What and Why?

- Similar to a list in indexing starting at 0
  - Can store any type of element
  - Can iterate over
- Immutable - Cannot mutate/change its value(s)
  - Efficient because it can't be altered
- Examples:
  - \( x = (5,6) \)
  - \( y = ([1,2],3.14) \)
Tuple Trace in Python Tutor

Print output (drag lower right corner to resize)

Frames       Objects

Global frame   tuple

Python 3.6
(known limitations)

1 x = (5, 6)
2 print(type(x))
3 y = (1, 2), 5, 3.14
4 y[0].append(8)
5 y[0][1] = 4
6 y[0] = [7, 9]

Print type

This part is immutable, cannot change any of it

This part is a list, which is mutable

8 was appended to the list, list is mutable

List element changed

Nothing is changed in the tuple

Still the address of the same list

Nothing is changed in the tuple
Variables and their Scope

- Local variable – variable in function only known in that function
- Parameter – way to pass information to a function
- Global variable - variable known throughout the whole file

When to use Global Variables

- Typically, don’t use global variables
  - Harder to share a function if it refers to a global variable
  - Act differently than other variables
- Sometimes makes sense
  - Global variable is used in most functions
  - Saves passing it to every function
- Best practice = help other humans read the code
  - Global variables define at top of file
  - When global used in function, declared as global at beginning of function

Summary - What is global?

- Accessible everywhere in the file (or “module”)
- Variable is in the global frame
  - First frame in Python Tutor
- If declared global in a function:
  - The variable in the global frame can also be reassigned in that function
  - Despite Python being in a different frame!
- Eliminates the need to pass this value to all the functions that need it
When reading code with globals

- When checking the value of a variable, ask:
  - Is this variable local to the function or in the global frame?
- When in a function and assigning a value to a variable, ask:
  - Has this variable been declared global?
    - If yes, reassign the variable in the **global frame**
    - If no, create/reassign the variable in the function's **local frame**

```
1  def func1():
2      s = "apple"
3      t = "plum"
4      print("func1 s: ", s, ", t: ", t)

5  def func2():
6      global s
7      s = "orange"
8      t = "grape"
9      print("func2 s: ", s, ", t: ", t)

10     if __name__ == '__main__':
11         print("main1 s: ", s)
12         t = "red"
13         print("main2 s: ", s, ", t: ", t)
14         func1()
15         print("main3 s: ", s, ", t: ", t)
16         func2()
17         print("main4 s: ", s, ", t: ", t)
```

Output:

```
main1 s: top
main2 s: red t: blue
func1 s: apple t: plum
func2 s: orange t: grape
main3 s: red t: blue
main4 s: orange t: blue
```
What will print?

Output:
main1 s: top
main2 s: red t: blue

Next call func1

What will print?

Output:
What will print?

Output:
main1 s: top
t: blue
func1 s: apple
t: plum
main3 s: red
t: blue
func2 s: orange
t: grape
main4 s: orange
t: blue

Notice t in main is always "blue"
s in main changed to "orange"

Now let's see the same thing in Python Tutor

- Global variables are in the global frame
Python Tutor – Step 9

2/17/22 Compsci 101, Spring 2022

Python 3.6

def func1():
    s = "apple"
    t = "plum"
    print("func1 s: ", s, " t: ", t)
    global s
    global t
    print("func1 s: ", s, " t: ", t)

def func2():
    global s
    s = 'orange'
    t = 'grape'
    print("func2 s: ", s, " t: ", t)
    global t
    t = 'blue'

if _name_ == '__main__':
    print('main1 s:', s)
    s = 'red'
    t = 'blue'
    print('main2 s:', s, " t: ", t)

func1()
func2()
print('main3 s:', s, " t: ", t)
func1()
func2()
print('main4 s:', s, " t: ", t)

Lines in main change global s

Next call func1

Python Tutor – Step 14

2/17/22 Compsci 101, Spring 2022

Python 3.6

def func1():
    s = "apple"
    t = "plum"
    print("func1 s: ", s, " t: ", t)

def func2():
    global s
    s = 'orange'
    t = 'grape'
    print("func2 s: ", s, " t: ", t)

if _name_ == '__main__':
    print('main1 s:', s)
    s = 'red'
    t = 'blue'
    print('main2 s:', s, " t: ", t)

func1()
func2()
print('main3 s:', s, " t: ", t)
func1()
func2()
print('main4 s:', s, " t: ", t)

s is local variable

s is global

There are two different s variables

Next call func1

Python Tutor – Step 16

2/17/22 Compsci 101, Spring 2022

Python 3.6

def func1():
    s = "apple"
    t = "plum"
    print("func1 s: ", s, " t: ", t)

def func2():
    global s
    s = 'orange'
    t = 'grape'
    print("func2 s: ", s, " t: ", t)

if _name_ == '__main__':
    print('main1 s:', s)
    s = 'red'
    t = 'blue'
    print('main2 s:', s, " t: ", t)

func1()
func2()
print('main3 s:', s, " t: ", t)
func1()
func2()
print('main4 s:', s, " t: ", t)

func1 did not change global s

Next call func2

func2 did not change s

Changed global s

No local s in func2
Python Tutor – Step 23

Variables
What, where, read, write? (in 101)

<table>
<thead>
<tr>
<th>What is it?</th>
<th>Where first created?</th>
<th>Where accessible? (read)</th>
<th>Where reassign-able? (write)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular variable in main</td>
<td>In main</td>
<td>In main only</td>
<td>In main only</td>
</tr>
<tr>
<td>(technically anywhere, but don’t do that)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Regular local function variable</td>
<td>In function</td>
<td>In function only</td>
<td>In function only</td>
</tr>
<tr>
<td>Global variable</td>
<td>Top of file</td>
<td>If not reassigning the value, in main and all functions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>In main or in any function that first declares it global</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Python will have an error if it is not declared global and it is used and then there is a variable with the same name being assigned.

Can avoid this by ALWAYS declaring the variable global in the function (best practice) if that is the variable you are using.

Assignment 3 Transform

- Uses several global variables.
- Only use global variables when we specify in an assignment.
Tuples

t = ([1], 2, 'three')
t[1] = 3
print(t[0][0])
print(type(t[0][0]))
t[0][0] = 4
print(t)
(x, y) = (t[1], t[0][0])
print(x, y)
x = t[1]
y = t[0][0]
print(x, y)
WOTO step through – step 7

```python
num = 0

def stuff(x):
    global num
    num += x
    return num

def thing(num):
    num += 1
    return num

if __name__ == '__main__':
    print('Beginning of main, num:', num)
    ret = stuff(5)
    print('After stuff num:', num, 'ret:', ret)
    ret = thing(10)
    print('After thing num:', num, 'ret:', ret)
```

Global num is 0

`x` is local inside function `stuff`

WOTO step through – step 10

```python
num = 0

def stuff(x):
    global num
    num += x
    return num

def thing(num):
    num += 1
    return num

if __name__ == '__main__':
    print('Beginning of main, num:', num)
    ret = stuff(5)
    print('After stuff num:', num, 'ret:', ret)
    ret = thing(10)
    print('After thing num:', num, 'ret:', ret)
```

Global num is 5

WOTO step through – step 11

```python
num = 0

def stuff(x):
    global num
    num += x
    return num

def thing(num):
    num += 1
    return num

if __name__ == '__main__':
    print('Beginning of main, num:', num)
    ret = stuff(5)
    print('After stuff num:', num, 'ret:', ret)
    ret = thing(10)
    print('After thing num:', num, 'ret:', ret)
```

WOTO step through – step 12

```python
num = 0

def stuff(x):
    global num
    num += x
    return num

def thing(num):
    num += 1
    return num

if __name__ == '__main__':
    print('Beginning of main, num:', num)
    ret = stuff(5)
    print('After stuff num:', num, 'ret:', ret)
    ret = thing(10)
    print('After thing num:', num, 'ret:', ret)
```
WOTO step through – step 13

```python
num = 0
def stuff(x):
global num
num += x
return num

def thing(num):
    num += 1
    return num

if __name__ == '__main__':
    print('Beginning of main, num: ', num)
    print('After stuff num: ', num, ', ret: ', ret)
    print('After thing num: ', num, ', ret: ', ret)
```

WOTO step through – step 16

```python
num = 0
def stuff(x):
global num
num += x
return num

if __name__ == '__main__':
    print('Beginning of main, num: ', num)
    print('After stuff num: ', num, ', ret: ', ret)
    print('After thing num: ', num, ', ret: ', ret)

    thing(num)
```

List `.index` vs String `.find`

```python
str = "computer"
pos = str.find("m")
pos = str.find("b")
lst = ["a", "b", "c", "a"]
indx = lst.index("b")
indx = lst.index("B")
```

Values:
m is 2
b is -1
indx is 1
ERROR, crash!
Use `.index` this way
Check if in!
List `.index` vs String `.find`

```python
str = "computer"
pos = str.find("m")  # m is 2
pos = str.find("b")  # b is -1

lst = ["a", "b", "c", "a"]
indx = lst.index("b")  # index is 1
indx = lst.index("B")  # ERROR, crash!
```

Values:
m is 2
b is -1

```python
lst = ["a", "b", "c", "a"]
indx = lst.index("b")  # index is 1
indx = lst.index("B")  # ERROR, crash!

indx = -1
if "B" in lst:
    indx = lst.index("B")  # Check if in!
```

Let's Write list Index function

- Call in `findIndex(lst, item)`
- Write it so it works like the string `find` function
  - `lst` is a list
  - `elm` is an element
  - Return the position of `elm` in `lst`
  - Return `-1` if `elm` not in `lst`
  - Use while loop to implement
- What is the while loop's Boolean condition?
  ```python
  index = 0
  while BOOL_CONDITION:
      index += 1
  ```

While Boolean condition

- What is the while loop's Boolean condition?
  ```python
  index = 0
  while BOOL_CONDITION:
      index += 1
  ```
While Boolean condition

```python
index = 0
while BOOL_CONDITION:
    index += 1
```

- What is the while loop's Boolean condition?
  - Whether found value: `lst[index] == elm`
  - Whether reach end of list: `index >= len(lst)`

DeMorgan's Law

- While loop stopping conditions, stop with either:
  - `lst[index] == elm`
  - `index >= len(lst)`
- While loop needs negation: DeMorgan's Laws
  - `not (A and B)` equivalent to `(not A) or (not B)`
  - `not (A or B)` equivalent to `(not A) and (not B)`

```python
while not (lst[index] == elm or index >= len(lst)):
    while lst[index] != elm and index < len(lst):
```

- Why did `==` become `!=`?
DeMorgan’s Law

• While loop stopping conditions, stop with either:
  • lst[index] == elm
  • index >= len(lst)

• While loop needs negation: DeMorgan’s Laws
  not (A and B) equivalent to (not A) or (not B)
  not (A or B) equivalent to (not A) and (not B)

while not (lst[index] == elm or index >= len(lst)):
  Why did >= become < ?
while lst[index] != elm and index < len(lst):

Think: DeMorgan’s Law

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>not (A and B)</th>
<th>(not A) or (not B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

Think: DeMorgan’s Law

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>not (A or B)</th>
<th>(not A) and (not B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

WOTO-2: Will this work?
Short Circuit Evaluation

- Short circuit evaluation, these are not the same!
  ```python
  while lst[index] != elm and index < len(lst):
      index += 1
  if index < len(lst):
      return index
  else:
      return -1
  ```

- As soon as truthiness of expression known
  - Stop evaluating
  - In \((A \text{ and } B)\), if \(A\) is false, do not evaluate \(B\)

Python Logic Summarized

- \(A \text{ and } B\) is True only when \(A\) is True and \(B\) is True
- \(A \text{ or } B\) is False only when \(A\) is False and \(B\) is False

<table>
<thead>
<tr>
<th>(A)</th>
<th>(B)</th>
<th>Evaluate (B) with and?</th>
<th>Evaluate (B) with or?</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Correct Code:

```python
def findIndex(lst, elm):
    index = 0
    while index < len(lst) and lst[index] != elm:
        index += 1
    if index < len(lst):
        return index
    else:
        return -1
```

Next look at future APT

- Understanding this APT will help you understand Assignment 3 Transform

APT - TxMsg

**Problem Statement**

Strange abbreviations are often used to write text messages on uncomfortable mobile devices. One particular strategy for encoding texts composed of alphabetic characters and spaces is the following:

- Spaces are maintained, and each word is encoded individually. A word is a consecutive string of alphabetic characters.

- If the word is composed only of vowels, it is written exactly as in the original message.

- If the word has at least one consonant, write only the consonants that do not have another consonant immediately before them. Do not write any vowels.

- The letters considered vowels in these rules are 'a', 'e', 'i', 'o' and 'u'. All other letters are considered consonants.

For instance, "ps i love u" would be abbreviated as "p i l v u" while "please please me" would be abbreviated as "ps ps m". You will be given the original message in the string parameter `original`. Return a string with the message abbreviated using the described strategy.

**Examples**

1. "text message"
   Returns "tx msg"

5. "aeiou bcdfghjklmnpqrstvwxyz"
   Returns: "aeiou b"
Debugging APTs: Going green

```python
def getMessage(original):
    ret = []
    for word in original.split():
        ret.append(transform(word))
    return " ".join(ret)
```

• TxMsg APT: from ideas to code to green
  • What are the main parts of solving this problem?
  • Transform words in original string
  • Abstract that away at first
  • Finding words in original string - .split()
  • Use another function `transform` to focus on one word
  • Then put list of words translated back together

Write helper function `transform`

• How?
• Use seven steps
• Work an example by hand

Transform word - Step 1: work small example by hand

• Word is “please”
• Letter is ‘p’, YES
• answer is “p”
• Letter is ‘l’, NO
• Letter is ‘e’, NO
• Letter is ‘a’, NO
• Letter is ‘s’, YES
• answer is “ps”
• Letter is ‘e’, NO
Step 2: Describe what you did
- Word is “please”, create an empty answer
- Letter is ‘p’, consonant, no letter before, YES
- Add ‘p’ to answer
- Letter is ‘l’, consonant, letter before “p”, NO
- Letter is ‘e’, vowel, letter before ‘l’, NO
- Letter is ‘a’, vowel, letter before ‘e’, NO
- Letter is ‘s’, consonant, letter before ‘a’, YES
- Add ‘s’ to answer
- Letter is ‘e’, vowel, letter before ‘s’, NO
- Answer is “ps”

Step 3: Find Pattern and generalize
Need to initialize letter before, pick “a”
answer is empty
for each letter in word
  If it is a consonant, and the letter before is a vowel, then add the letter to the answer
  This letter is now the letter before
return answer

Step 4 – Work another example
- Word is message
- Letter is ‘m’, before is ‘a’, add ‘m’ to answer
- Letter is ‘e’, before is ‘m’, NO
- Letter is ‘s’, before is ‘e’, add ‘s’ to answer
- Letter is ‘s’, before is ‘s’, NO
- Letter is ‘a’, before is ‘s’, NO
- Letter is ‘g’, before is ‘a’, add ‘g’ to answer
- Letter is ‘e’, before is ‘g’, NO
- Answer is “msg”  WORKS!!

Step 5: Translate to Code
# Letter before is “a”  # start with a vowel

# answer is empty

# for each letter in word
Step 5: Translate to Code

# Letter before is “a”       # start with a vowel
before = ‘a’

# answer is empty
answer = []          # or this could be an empty string

# for each letter in word
for ch in word:

    #If it is a consonant, and the letter before is a #vowel, then add the letter to the answer
    if !(isVowel(ch)) and isVowel(before):
        answer += ch

    #This letter is now the letter before
    before = ch

# return answer
return answer

STOP HERE…

• You finish
• May need to debug
Why use helper function 'transform'?

• **Structure of code is easier to reason about**
  • Harder to develop this way at the beginning
  • Similar to accumulate loop, build on what we know

• **We can debug pieces independently**
  • What if transform returns "" for every string?
  • Can we test transform independently of getMessage?