DeMorgan’s Law, Short circuiting, Global, Tuples

Susan Rodger
October 13, 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”

“We take large challenges and turn them into opportunities that will one day help people and awaken the problems that can be solved. We believe in setting new standards for what is possible.”
Announcements

• APT-3 due tonight
• Assign 3 due Thursday, Oct 20
  • Sakai Assign 3 quiz due Tues. Oct 18 (no grace day!)
• Lab 5 on Friday, do prelab
• Exam 1 handed back on Gradescope
  • Regrades through Oct 17, go to problem in gradescope and request a regrade for that problem
• Midterm grades on Dukehub – rough estimate!

• APT Quiz 1 – Oct 13-17
PFTD

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

```python
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 output (drag lower right corner to resize)

Frames          Objects
Tuple Trace in Python Tutor

Python 3.6 (known limitations)

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

Python 3.6 (known limitations)

1. \( x = (5, 6) \)
2. \( \text{print(type(x))} \)
3. \( y = ([1, 2], 5, 3.14) \)
4. \( y[0].\text{append}(8) \)
5. \( y[0][1] = 4 \)
6. \( y[0] = [7, 9] \)
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
What is a global variable?

- 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 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
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**
s = 'top'

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()
    print('main3 s:', s, "t:", t)
    func2()
    print('main4 s:', s, "t:", t)
Now let’s see the same thing in Python Tutor

• Global variables are in the global frame
```python
Python 3.6
(known limitations)

1 s = 'top'

2

3 def func1():
4     s = "apple"
5     t = "plum"
6     print("func1 s:", s, "t:", t)
7
8 def func2():
9     global s
10    s = 'orange'
11    t = 'grape'
12    print('func2 s:', s, "t:", t)
13
14 if __name__ == '__main__':
15    print('main1 s:', s)
16    s = 'red'
17    t = 'blue'
18    print('main2 s:', s, "t:", t)
19    func1()
20    print('main3 s:', s, "t:", t)
21    func2()
```
# Variables

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular variable in main</td>
<td>In main</td>
<td>In main only (technically anywhere, but don’t do that)</td>
<td>In main only</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>In main or in any function that first declares it global</td>
</tr>
</tbody>
</table>
Assignment 3 Transform

• Uses several global variables.

• Only use global variables when we specify in an assignment
WOTO-1 – Tuples and Globals
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!
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?
  ```
  index = 0
  while BOOL_CONDITION:
    index += 1
  ```
While Boolean condition

index = 0
while BOOL_CONDITION:
    index += 1

- What is the while loop’s Boolean condition?
DeMorgan’s Law

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

• While loop needs negation: DeMorgan's Laws
  \(
  \text{not (A and B)} \text{ equivalent to (not A) or (not B)}
  \)
  \(
  \text{not (A or B)} \text{ equivalent to (not A) and (not B)}
  \)
Think: DeMorgan’s Law

<table>
<thead>
<tr>
<th>A</th>
<th>B</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></td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</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></td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td></td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td></td>
<td>True</td>
</tr>
</tbody>
</table>
WOTO-2: Will this work?


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

• As soon as truthiness of expression known
  • Stop evaluating
  • In (A and B), if A is false, do not evaluate B
Python Logic Summarized

• A and B is True only when A is True and B is True
• A or B is False only when A is False and B is False

• Short-circuit evaluation of A or B?
  • If A is true, do not evaluate B

<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>
APT Quiz 1 Oct 13-17

• Opens 10/13 Noon
• Closes at 11pm 10/17 – must finish all by this time
• There are two parts based on APTs 1-3
  • Each part has two APT problems
  • Each part is 2 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
• 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 online between 9pm and 9am!
  • We are not on all the time, especially weekends
  • Will try to answer questions between 9am – 9pm
    • About typos, cannot help you in solving APTs

• See 101 APT page for tips on debugging APTs
Don't go to Sakai to start APT Quiz until you are ready to start

If you click on it, you start it!
Tips for APT Quiz

• Don't like the format, convert it:

  • dig = "458"     Is variable dig a number?

• Use 7 steps
Tips for APT Quiz

• Write a helper function

• Break code into parts
Problem 1

• Write function `addto`. Given `wordlist`, a list of words and `numlist`, a list of numbers, return new list with a number from `numlist` attached to the end of each string. Repeat numbers from `numlist` if you need more numbers

  • `numlist = [3, 5, 6]`
  • `wordlist = ["on", "to", "a", "be", "some", "fa", "so"]`
  • Result: ["on3", "to5", "a6", "be3", "some5", "fa6", "so3"]

• How to solve:
WOTO-3: function addto
def addto(wordlist, numlist):

nlist = numlist
answer = []
if len(numlist) < len(wordlist):
    nlist = numlist * len(wordlist)    # plenty big
for index in range(len(wordlist)):
    answer.append(wordlist[index] + str(nlist[index]))
return answer
Problem 2

• Write function update that has one parameter, a list of numbers and/or words.

• This function makes a new list by starting with the original list and adds 1 to each number in the list. The string returned is the sum of the modified numbers in the list, a colon, followed by the elements in the modified list, separated by a dash.

• Example:
  • update([1, 5, 'a', 2, 'z']) returns "11:2-6-a-3-z"
  • update([87, 'car', 11, 'be']) returns "100:88-car-12-be"
How to solve

• For each element in list, is it a number?
• For numbers only add 1
• Sum only numbers, avoid strings
• Convert numbers to strings to build final string
def update(alist):

    onemore = []
    for item in alist:
        if str(item)[0] in "0123456789":
            onemore.append(item+1)
        else:
            onemore.append(item)

    total = 0
    for x in onemore:
        if str(x)[0] in "0123456789":
            total += x

    final = [str(x) for x in onemore]
    return str(total) + ":" + "-".join(final)