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

Susan Rodger
February 23, 2023

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, March 2
  - Sakai Assign 3 quiz due Tues. Feb 28 (no grace day!)
- Lab 6 on Friday, do prelab
- Midterm grades coming – rough estimate!
- APT Quiz 1 – Feb 23 (today 1pm) – Mon, Feb 27
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:
  - \( x = (5, 6) \)
  - \( y = ([1, 2], 3.14) \)
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
Nothing is changed in the tuple
Still the address of the same list

List element changed
Nothing is changed in the tuple

Can't change any element in the tuple
ERROR if you try to change any part of 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

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
Global

What will print?

Local variable s

What will print?

Use global s

What will print?

Output:

main1 s: top
main2 s: red t: blue
func1()
func2()
func1()
func2()
func1()
func2()
Next call func1
What will print?

Output:

main1 s: top
t: blue

main2 s: red
t: blue

func1 s: apple
t: plum

main3 s: red
t: blue

func2 s: orange
t: grape

main4 s: orange
t: blue

Next call func2
What will print?

Output:
main1 s: top
t: red
func1 s: apple
t: plum
main2 s: red
t: blue
func2 s: orange
t: grape
main3 s: red
t: blue
func3 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
There are two different s variables

s is local

s is global

func1 did not change global s

Next call func2

Changed global s

No local s in func2

Change to s in func2 permanent

s is local variable
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</td>
<td>In main</td>
<td>In main only</td>
<td>In main only (technically anywhere, but don’t do that)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>In main only</td>
</tr>
<tr>
<td>Regular local function</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>

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

WOTO-1 – Tuples and Globals
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)

Notice there is NO variable assigned. There is no z = (x,y). This is a way to assign two variables at the same time. We are creating x and y both on the same line as new variables and giving them values.

Similar!

WOTO step through – step 6

```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 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

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)
    ret = stuff(5)
    print('After stuff num:', num, 'ret:', ret)
    ret = thing(10)
    print('After thing num:', num, 'ret:', ret)
```

**Update:**
- Updated local num
- Global num unchanged

WOTO step through – step 16

```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 = thing(10)
    print('After thing num:', num, 'ret:', ret)
```

**Values:**
- m is 2
- b is -1
- indx is 1

ERROR, crash!
Use `.index` this way.
Check if in!

WOTO step through – last step

```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)
```

**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:**
- pos = 2
- pos = -1
- indx = 1
- indx = 1
List `.index` vs String `.find`

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

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

Values:
pos is 2  pos is -1
indx is 1  ERROR, crash!

lst.index(item) program crashes if item is not there!

Let's Write list Index function

• Call in findIndex(lst, elm)
• 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
    if "B" in lst:
        indx = lst.index("B")
        ERROR, crash!
    if "B" in lst:
        indx = lst.index("B")
        ERROR, crash!
```
While Boolean condition

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

- What is the while loop’s stopping 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)):
    # equivalent to: not (lst[index] == elm and 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 lst[index] != elm and index < len(lst):
    # equivalent to: lst[index] != elm and 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)

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

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>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>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

Fill in the blanks

<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>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>

Why did == become != ?

Why did >= become < ?

Think: DeMorgan’s Law

<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>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>

Fill in the blanks

<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>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

Why did >= become < ?
Short Circuit Evaluation

• Short circuit evaluation, these are not the same!

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

```python
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
```
Short Circuit Evaluation

- Short circuit evaluation, these are not the same!
  - First condition depends on second condition
  
  ```python
  while lst[index] != elm and index < len(lst):
    while index < len(lst) and lst[index] != elm:
  ```

- 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
  - If A is True
    - Need to evaluate B
  - If A is False
    - Don't need to evaluate B

- A or B is True if one of A and B are True
  - if A is True
    - Don’t need to evaluate B
  - If A is False
    - Need to evaluate B

- Short-circuit evaluation A and B, A or B
  - If evaluation of A gives you the answer, you don’t need to evaluate B

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
```
APT Quiz 1 Feb 23-27

- Opens 2/23 1pm
- 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 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:
  - "lots of words" → ["lots", "of", "words"]
  - "6 3 9" → ['6', '3', '9'] → [6, 3, 9]

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

- Use 7 steps

Tips for APT Quiz

- Write a helper function

- Break code into parts

Tips for APT Quiz

- Don't like the format, convert it:

- Use 7 steps
  - Work an example by hand
  - Code – what do you need? Loop over what? If?

Tips for APT Quiz

- Write a helper function
  - What if had function to do X?
    - Test function before you use it
  - If you have a loop inside a loop
    - Instead put the inside loop in a function and call it
    - Simplifies your code
    - Easier to debug

- Break code into parts
  - Do one part at a time
  - Print values of variables for each part
  - You think it does one thing, You might be surprised
Problem 1

- Write function `addto`. Given `wordlist`, a list of words and `numlist`, a list of integers, return a new list with a number from `numlist` in the same position attached to the end of each string. Repeat numbers from `numlist` in the same order 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:
  - Loop through `numlist` multiple times – TRICKY!
  - Easier: create "new" `numlist` that is longer
    - Create `nlist` is [3, 5, 6, 3, 5, 6, 3, 5, 6]
    - Use a for loop to do this
    - OR: `nlist = numlist*3`

Let's solve!

- Make list of numbers long enough
- Use indexing
  - Index into `wordlist` and same position in `numlist`
- Use a loop over `wordlist` and create a new list
  - Accumulation pattern!
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 integers 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

```python
def update(alist):
    onemore = []
    for item in alist:
        if str(item)[0] in "0123456789":  # just check 1st digit
            onemore.append(item + 1)  # add 1 to number
        else:
            onemore.append(item)  # add word
    total = 0
    for x in onemore:
        if str(x)[0] in "0123456789":  # if it is a number
            total += x
    final = [str(x) for x in onemore]  # convert all to strings
    return str(total) + ":" + "-".join(final)
```