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]

Frames     Objects
```

Print output (drag lower right corner to resize)
Tuple Trace in Python Tutor

```python
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]
```
Tuple Trace in Python Tutor

```
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]
```

- This part is immutable, cannot change any of it
- This part is a list, which is mutable
- Print type

This part is immutable, cannot change any of it

10/13/22 Compsci 101, Fall 2022
Tuple Trace in Python Tutor

Python 3.6
(know 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)
<class 'tuple'>

8 was appended to the list, list is mutable

Still the address of the same list

Nothing is changed in the tuple
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. \( y[0].append(8) \)
5. \( y[0][1] = 4 \)
6. \( y[0] = [7,9] \)

Edit this code

- line that just executed
- next line to execute

Print output (drag lower right corner to resize)

<class 'tuple'>

Frames

Objects

Global frame

List element changed

Nothing is changed in the tuple
Tuple Trace in Python Tutor

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]

Can’t change any element in the tuple

ERROR if you try to change any part of the tuple

Types

Global frame

x

y

Frames

Objects

tuple

0 1
5 6

list

0 1 4 8

tuple

0 1 5 3.14

TypeError: 'tuple' object does not support item assignment

Edit this code

line that just executed

next line to execute

Done running (6 steps)
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
```python
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)
```
```python
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)
```

What will print?

Output:

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

```
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)
```

Output:
main1 s: top
1. `s = 'top'

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

3. **def** func2():
   4. `global s`
   5. `s = 'orange'
   6. `t = 'grape'
   7. `print("func2 s: ", s, "t: ", t)

8. **if** __name__ == '__main__':
   9. `print('main1 s: ', s)
   10. `s = 'red'
   11. `t = 'blue'
   12. `print('main2 s: ', s, "t: ", t)
   13. `func1()
   14. `print('main3 s: ', s, "t: ", t)
   15. `func2()
   16. `print('main4 s: ', s, "t: ", t)
What will print?

Output:
main1 s: top
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
```python
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)
```

What will print?

Output:
main1 s: top
main2 s: red t: blue
func1 s: apple t: plum
main3 s: red t: blue

Next call func2
What will print?

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

```
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)
```
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)

Output:
main1 s: top
func1 s: apple t: plum
main2 s: red t: blue
func2 s: orange t: grape
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
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()
```
Python Tutor – Step 9

Python 3.6
(known limitations)

```python
2 def func1():
3     s = "apple"
4     t = "plum"
5     print("func1 s:", s, "t:", t)
6     global s
7     global t
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()
22    print('main4 s:', s, "t:", t)
```

Lines in main change global s

Next call func1
Python 3.6
(known limitations)

```python
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()

main1 s: top
main2 s: red t: blue
func1 s: apple t: plum
```

There are two different `s` variables:
- `s` is global
- `s` is local variable
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()
Python Tutor – Step 21

```python
import known limitations

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)
```

Print output (drag lower right corner to resize):

```plaintext
main1 s: top
main2 s: red t: blue
func1 s: apple t: plum
main3 s: red t: blue
func2 s: orange t: grape
```

Frames

Objects

Global frame
```
    s  "orange"
    t  "blue"
```

function

func1()

function

func2()

No local s in func2

Changed global s
```python
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)
```

Change to s in func 2 permanent
# 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 (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>
## 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 (technically anywhere, but don’t do that)</td>
<td>In main only</td>
</tr>
<tr>
<td>in main</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Regular local</td>
<td>In function</td>
<td>In function only</td>
<td>In function only</td>
</tr>
<tr>
<td>function variable</td>
<td></td>
<td></td>
<td></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>

### Notes:

- 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.

10/13/22 Compsci 101, Fall 2022
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)
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)  
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.
t = ([1], 2, 'three')

print(t[0][0])
print(type(t[0][0]))

print(t[0][0] = 4)
print(t)

(x, y) = (t[1], t[0][0])
print(x, y)
print((x, y))
x = t[1]
y = t[0][0]
print(x, y)
WOTO step through – step 6

Python 3.6
(known limitations)

```python
1 num = 0
2
3 def stuff(x):
4     global num
5     num += x
6     return num
7
8 def thing(num):
9     num +=1
10    return num
11
12 if __name__ == '__main__':
13     print('Beginning of main, num:', num)
14     ret = stuff(5)
15     print('After stuff num:', num, 'ret:', ret)
16     ret = thing(10)
17     print('After thing num:', num, 'ret:', ret)
```

Print output (drag lower right corner to resize)

Beginning of main, num: 0

Frames          Objects

Global frame
    num  0
    stuff
    thing

function stuff(x)
function thing(num)
WOTO step through – step 7

Python 3.6
(known limitations)

```python
1 num = 0
2
3 def stuff(x):
4     global num
5     num += x
6     return num
7
8 def thing(num):
9     num += 1
10    return num
11
12 if __name__ == '__main__':
13    print('Beginning of main, num:', num)
14    ret = stuff(5)
15    print('After stuff num:', num, 'ret:', ret)
16    ret = thing(10)
17    print('After thing num:', num, 'ret:', ret)
```

Print output (drag lower right corner to resize)

Beginning of main, num: 0

Frames
Objects

Global frame

num 0
stuff
thing

stuff
x 5

Global num is 0
num is global num
x is local inside function stuff
WOTO step through – step 10

```python
Python 3.6
(known limitations)

1 num = 0
2
3 def stuff(x):
4     global num
5     num += x
6     return num
7
8 def thing(num):
9     num += 1
10    return num
11
12 if __name__ == '__main__':
13    print('Beginning of main, num:', num)
14    ret = stuff(5)
15    print('After stuff num:', num, 'ret:', ret)
16    ret = thing(10)
17    print('After thing num:', num, 'ret:', ret)
```

Global num is 5
WOTO step through – step 11

```
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)
```

Print output (drag lower right corner to resize)
Beginning of main, num: 0

Frames
Global frame
num 5
stuff
thing
ret 5

Objects
function stuff(x)
function thing(num)
WOTO step through – step 12

Python 3.6

```python
(known limitations)

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)

print output (drag lower right corner to resize)
Beginning of main, num: 0
After stuff num: 5 ret: 5
```

Frames

- Global frame
  - num: 5
  - stuff
  - thing
  - ret: 5

Objects

- function stuff(x)
- function thing(num)
WOTO step through – step 13

```python
Python 3.6
(known limitations)

num = 0

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

beginning of main, num: 0

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

frame of stuff(x)
frame of thing(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)
```

beginning of main, num: 0
After stuff num: 5 ret: 5

Frames

Objects

Global frame

- num 5
- stuff
- thing
- ret 5

thing

num 10

num is local variable
num is local variable
WOTO step through – step 16

```python
1  num = 0
2
3  def stuff(x):
4      global num
5      num += x
6      return num
7
8  def thing(num):
9      num +=1
10     return num
11
12 if __name__ == '__main__':
13     print('Beginning of main, num:', num)
14     ret = stuff(5)
15     print('After stuff num:', num, 'ret:', ret)
16     ret = thing(10)
17     print('After thing num:', num, 'ret:', ret)
```

Print output (drag lower right corner to resize)

- Beginning of main, num: 0
- After stuff num: 5 ret: 5
- Frames
  - Global frame
    - num 5
    - stuff
    - thing
  - Return
    - num 11
- Objects
  - function stuff(x)
  - function thing(num)
  - Updated local num
  - Global num unchanged
WOTO step through – last step

```python
Python 3.6
(known limitations)

1  num = 0
2
3  def stuff(x):
4      global num
5      num += x
6      return num
7
8  def thing(num):
9      num +=1
10     return num
11
12 if __name__ == '__main__':
13    print('Beginning of main, num:', num)
14    ret = stuff(5)
15    print('After stuff num:', num, 'ret:', ret)
16    ret = thing(10)
17    print('After thing num:', num, 'ret:', ret)
```
List .\texttt{index} vs String .\texttt{find}

\texttt{str} = "\texttt{computer}" \\
\texttt{pos} = \texttt{str.find("m")} \\
\texttt{pos} = \texttt{str.find("b")}

\texttt{lst} = ["a", "b", "c", "a"] \\
\texttt{indx} = \texttt{lst.index("b")} \\
\texttt{indx} = \texttt{lst.index("B")}

Values:

m is 2
b is -1
indx is 1

ERROR, crash!

Use .\texttt{index} this way

Check if in!
List `.index` vs String `.find`

```python
str = "computer"
pos = str.find("m")
pos = str.find("b")
```

```python
lst = ["a", "b", "c", "a"]
indx = lst.index("b")
indx = lst.index("B")
```

Values:
- `m` is 2
- `b` is -1

```python
indx is 1
ERROR, crash!
```

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

indx = -1
if "B" in lst:
    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, 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?
  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?
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
  
  \[
  \text{not } (A \text{ and } B) \text{ equivalent to } (\text{not } A) \text{ or } (\text{not } B) \\
  \text{not } (A \text{ or } B) \text{ equivalent to } (\text{not } A) \text{ and } (\text{not } B)
  \]
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)):
  Is equivalent to: (not A) or (not B)

while lst[index] != elm and index < len(lst):
DeMorgan’s Law

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

• While loop needs negation: DeMorgan's Laws
  \[
  \neg (A \text{ and } B) \text{ equivalent to } (\neg A) \text{ or } (\neg B)
  \]
  \[
  \neg (A \text{ or } B) \text{ equivalent to } (\neg A) \text{ and } (\neg B)
  \]

while not (\text{lst}[\text{index}] == \text{elm} \text{ or } \text{index} >= \text{len}(	ext{lst})):  

While \( \text{lst}[\text{index}] != \text{elm} \) and \( \text{index} < \text{len}(	ext{lst}) \):

Why did == become != ?
DeMorgan’s Law

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

• While loop needs negation: DeMorgan's Laws
  \( \neg (A \land B) \) equivalent to \( (\neg A) \lor (\neg B) \)
  \( \neg (A \lor B) \) equivalent to \( (\neg A) \land (\neg B) \)

while not (\text{lst}[\text{index}] == \text{elm} \lor \text{index} \geq \text{len(lst)}):

Why did \( \geq \) become \( < \) ?

while \text{lst}[\text{index}] != \text{elm} \land \text{index} < \text{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></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></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>False</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>True</td>
<td>True</td>
</tr>
</tbody>
</table>
Think: DeMorgan’s Law

**Fill in the blanks**

<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>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</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>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?

```python
6 def findIndex(lst, elm):
7     index = 0
8     while lst[index] != elm and index < len(lst):
9         index += 1
10     if index < len(lst):
11         return index
12     else:
13         return -1
```
WOTO-2: Will this work?
Short Circuit Evaluation

• Short circuit evaluation, these are not the same!

```
while lst[index] != elm and index < len(lst):
    # Code
```

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

• Short circuit evaluation, these are not the same!

  First condition depends on second condition

while lst[index] != elm and index < len(lst):

  Put second condition first!

while index < len(lst) and lst[index] != elm:

• 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 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>
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 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
APT Quiz

Start the APT quiz on Sakai under quizzes, but not until you are ready to take the quiz.

APT Tips

See below for hints on what to do if your APT doesn't run.

For each problem in an APT set, complete these steps by the due date:

- first click on the APT set below to go to the APT page.
- write the code, upload the file and click the Submit link
- check your grade on the grade code page by clicking on check submissions

In solving APTs, your program should work for all cases, not just the test cases we provide. We may test your program on inputs we did not provide.

<table>
<thead>
<tr>
<th>APT</th>
<th>Due Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>APT-1</td>
<td>Sept. 15</td>
</tr>
<tr>
<td>APT-2</td>
<td>Sept. 29</td>
</tr>
<tr>
<td>APT-3</td>
<td>Oct. 13</td>
</tr>
<tr>
<td>PRACTICE FOR APT QUIZ 1</td>
<td>PRACTICE ONLY</td>
</tr>
</tbody>
</table>

We may do some APTs partially in class or lab, but you still have to do them and submit them. There will usually be extra points you can do more than required to challenge yourself. We do notice if you do more APTs than those required. If you do extra, you still have to be turned in on the due date.

Regrades

If you have concerns about an item that was graded (lab, apt or assignment), you have one week after the grade is posted to fill out a regrade form here.

Problems Running an APT? Some Tips!

Stuck! Use 7 steps!
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

• 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?
  • Is each letter in "0123456789"?
  • For ch in dig:
    • if ch not in "0123456789"
    •   # not a digit!

• 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

• Break code into parts
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 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
Problem 1

• Write function addto. Given list of words and list of integers, return new list with one number to end of each string, repeat numbers 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!
Practice for APT Quiz 1

def addto(wordlist, numlist):

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
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)
def update(alist):
    onemore = []
    for item in alist:
        if str(x)[0] in "0123456789":  # just check 1st digit
            onemore.append(x+1)  # add 1 to number
        else:
            onemore.append(x)  # 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)