Compsci 101
List Comprehensions, Transform, Global
Part 1 of 2

Susan Rodger

September 29, 2020

```
ret = [V_EXP for V in LIST if BOOL_EXP]
```

PFTD

- List Comprehensions
- Global Variables
- Transform Assignment

Accumulator in one line

```
def onlyPos(nums):
    ret = []
    for n in nums:
        if n > 0:
            ret.append(n)
    return ret

print(onlyPos([1,2,3,-1,-2,-3]))
```

```
return [n for n in nums if n > 0]
```

- List Comprehension
  - We will use a complete, but minimal version of list comprehensions, much more is possible

List Comprehension Syntax

```
ret = []
for V in LIST:
    ret.append(V_EXP)
```

```
ret = [V_EXP for V in LIST]
```

```
ret = []
for V in LIST:
    if BOOL_EXP:
        ret.append(V_EXP)
```

```
ret = [V_EXP for V in LIST if BOOL_EXP]
```

- **V** is any variable: all list elements in order
- **V_EXP** is any expression, often use **V**

List Comprehension Syntax

```
ret = []
for V in LIST:
    ret.append(V_EXP)
```

```
ret = [V_EXP for V in LIST]
```

```
ret = []
for V in LIST:
    if BOOL_EXP:
        ret.append(V_EXP)
```

```
ret = [V_EXP for V in LIST if BOOL_EXP]
```

- **`if`** part optional - **`BOOL_EXP`** is a Boolean expression usually using **`V`**

List Comprehension Examples

```
print( [n*2 for n in range(10)] )

print( [n for n in range(10) if n % 2 == 1] )

print( [n/2 for n in range(10) if n % 2 == 0] )

lst = [‘banana’, ‘pineapple’, ‘apple’]
print( [c for c in lst if ‘n’ in c] )
```

Compsci 101
List Comprehensions, Transform, Global
Part 2 of 2

Susan Rodger

September 29, 2020

```
ret = [V_EXP for V in LIST if BOOL_EXP]
```

Global Variables (Best Practice)

- Best practice = help other humans read the code

- All variables that will be global are created with an initial assignment at the top of the file
- When used in a function, variable is declared global at the beginning of the function

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**

What will this print?

```
1   s = 'top of the file'
2
3   def change():
4       global s
5       s = 'I am'
6       t = 'changing'
7       print('Inside change:', s, t)
8
9
10  if __name__ == '__main__':
11      print('Beginning of main:', s)
12      s = 'inside main'
13      t = 'text'
14
15      print('Before change:', s, t)
16      change()
17      print('After change:', s, t)
```

```
1   s = 'top of the file'
2
3   def change():
4       global s
5       s = 'I am'
6       t = 'changing'
7       print('Inside change:', s
8
9
10  if __name__ == '__main__':
11      print('Beginning of main:', s)
12      s = 'inside main'
13      t = 'text'
14
15      print('Before change:', s, t)
16      change()
17      print('After change:', s, t)
```

Remember: When considering a variable ask is this variable global?
- If yes: look in global (first) frame for value AND when reassigning do it in the global frame

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

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

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