

# Lecture 11

4.5 List Comprehension

5.1 While loops

5.5 Error Checking and Exceptions

## 4.5 List Comprehension

Assume that we have a list of people invited for a party:

```
guests = ['Tom', 'Mary', 'Lily', 'James']
```

Compare the following two pieces of code:

```
guests_lc = []  
for person in guests:  
    guests_lc.append(person.lower())
```

and

```
guests_lc = [person.lower() for person in guests]
```

**Syntax:**

```
result = [expression for identifier in sequence]
```

## 4.5 List Comprehension

### Syntax:

*result = [expression **for** identifier **in** sequence]*

### More **General Syntax**:

*result = [expression **for** identifier **in** sequence **if** condition]*

it is the same as:

```
result = []  
for identifier in sequence:  
    if condition:  
        result.append(expression)
```

## 4.5 List Comprehension

**Example:** exercise 4.38 on page 156

Given a list **orig**, possibly containing duplicate values, show how to use *list comprehension* to produce a new list **uniq** that has all values from the original but with duplicated omitted.

*Hint:* look for indices at which the leftmost occurrence of a value occurs.

## 4.5 List Comprehension

**Example:** exercise 4.38 on page 156

Given a list **orig**, possibly containing duplicate values, show how to use *list comprehension* to produce a new list **uniq** that has all values from the original but with duplicated omitted.

*Hint:* look for indices at which the leftmost occurrence of a value occurs.

```
uniq = [item for item in orig if item not in uniq]
```

## 4.5 List Comprehension

**Example:** exercise 4.38 on page 156

Given a list **orig**, possibly containing duplicate values, show how to use *list comprehension* to produce a new list **uniq** that has all values from the original but with duplicated omitted.

*Hint:* look for indices at which the leftmost occurrence of a value occurs.

```
uniq = [item for item in orig if item not in uniq]
```

- is not working

How about this one? :

```
l=len(orig)  
uniq=[orig[i] for i in range(l) if orig[i] not in orig[i+1:l]]
```

## 4.5 List Comprehension

See programs `exercise_4-38.py`,  
`exercise_4-38-mod1.py`, and  
`exercise_4-38-mod2.py`

# 5.1 While loops

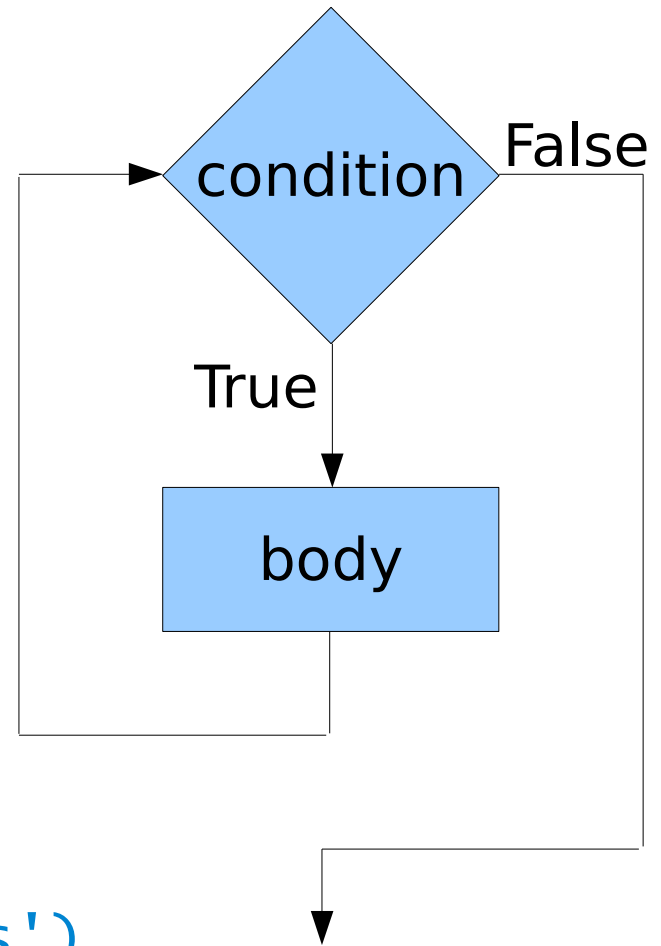
## Syntax:

```
while condition:  
    body
```

! In any loop body the command **break** causes an immediate stop to the entire loop.  
(very useful sometimes)

## Example:

```
while response.lower() in ('y', 'yes')  
    # body of the loop:  
    ...  
    response=raw_input("would you like to continue (Yes/No)?")  
  
# out of the loop code:  
...
```



# 5.1 While loops

## Infinite loops

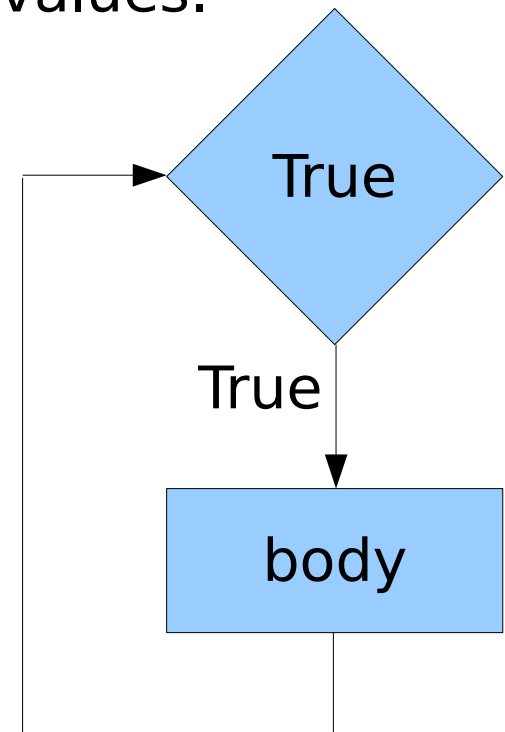
When working with a for loop, the overall number of iterations is naturally bounded based on the length of the original sequence.

When working with a while loop, the overall number of iterations is not explicitly bounded. It is determined by a combination of the loop condition and the changing state of underlying values.

This introduces a potential pitfall: a possibility that the while loop never ends (**infinite loop**)

### Example:

```
while True:  
    Print "Hello!"
```



# 5.1 While loops

**Example:** exercise 5.16 on page 198

An integer  $k \geq 2$  is a **prime number** if it is not evenly divisible by any numbers in **range(2,k)**. Build a list of all prime numbers less than or equal to a value **n** entered by the user.

*Hint:* For each  $2 \leq k \leq n$ , test its primality by looping over `range(2,k)` looking for a factor of  $k$

*One more comment:*

When checking whether a number, say  $k$ , is prime or not, there is no need to check the divisibility of this number by a number, whose square is more than  $k$ .

- this reduces the complexity of an algorithm.

**Question:** what numbers we don't need to check for primality?

# 5.1 While loops

```
n=input("Please, enter a whole number > 1:")  
  
# list of primes initially consists of one value  
primes = [2]  
  
for k in range(3,n+1):  
    d=2 # divisor  
  
    while k%d > 0 and d**2 < k:  
        d+=1  
  
    if k%d != 0:  
        primes.append(k)  
  
    k += 1 # skipping even numbers  
  
print primes
```

# Homework Assignment

## HW7:

Exercises 4.35,  
4.39 (bound the highest possible number for the user to  
enter to 9),  
5.9