Looping

Looping (recap)

- Branching allows the program to execute certain statements if certain conditions are met;
- Loops execute certain statements while certain conditions are met.
- Python has two kinds of loops:
  - While
  - for

Definite Loop using `for`

- Sometimes you want to loop through a set of items
  - a list of strings
  - a list of numbers

- We can use `for` loop to iterate through a set of things
  - `for` loop is called *definite* loop
  - `while` loop is called *indefinite* loop (it loops until some condition becomes `False`)

`for` - loop

- Syntax:

```
for item in sequence:
   statement
   statement
   statement
```

- **For loops iterate over a given sequence.**
- **`for` and `in` are keywords**
- **: is necessary**
- **`item` is an arbitrary variable name**
- **block of statements should be indented**

- The block of lines is repeated once for each element of the sequence,
  - in each iteration `item` takes the next value in the sequence

`for` - loop

- The `for` loop iterates over a given sequence

```
for count in [1,2,3,4,5]:
   print count, count*count
```

```
1 1
2 4
3 9
4 16
5 25
```

- in each iteration `count` takes the next value in the list

`list` Type

- Lists are ordered sequences of arbitrary data.
- Lists are the first kind of data discussed so far that are mutable:
  - the length of the sequence can be changed
  - elements can be substituted

The basic format:
- **A square-bracket-enclosed,**
- **comma-separated list of arbitrary data.**

```
[1,2,3,4,5]
["red","blue","green"]
["silly", 44, "mixed", -2, "green"]
[]  #empty list
```

`for` - loop

- The `for` loop iterates over a given sequence

```
for color in ["red","blue","green"]:
   print color
```

```
red
blue
green
```

- in each iteration `color` takes the next value in the list

`for` - loop

- The `for` loop iterates over a given sequence

```
primes = [2,3,5,7]
for var in primes:
   print var
```

```
2
3
5
7
```

- `primes` is a list of integer numbers

for - loop

- The `for` loop iterates over a given sequence

```
friends = ["Joseph","Glenn","Sally"]
for x in friends:
   print “Happy new year ”,x


print “Done!”
```

```
Happy new year Joseph
Happy new year Glenn
Happy new year Sally
Done!
```

`for` - loop

- The `for` loop iterates over a given sequence

```
friends = []
for x in friends:
   print “Happy new year ”,x


print “Done!”
```

`for` - loop

- The `for` loop iterates over a given sequence

```
friends = []
for x in friends:
   print “Happy new year ”,x


print “Done!”
```

```
Done!
```

- `[]` is an empty list, so this loop has no iteration

`for` - loop

- The `for` loop iterates over a given sequence

```
numbers = [1,10,20,30,40,50]
sum = 0
for num in numbers:
    sum += num

print sum
```

```
151
```

- Compute the summation of a list of integer numbers

`range()` Function

- There is a built-in function `range`, that can be used to automatically generate regular arithmetic sequences.

- The general pattern to use is:

```
range(sizeOfSequence)
```

- Try this in the shell:

```
>>> range(4)

>>> range(10)
```

- `range(n)` generates a sequence of integers starting at 0 (ends before `n`)
  - [0, 1, 2, …, n-1] #a sequence of n elements

`range()` Function

- There is a built-in function `range`, that can be used to automatically generate regular arithmetic sequences.

- The general pattern to use is:

```
range(sizeOfSequence)
```

- Try this in the shell:

```
>>> range(4)

>>> range(10)
```

The `range` function can be used to generate a much wider variety of sequences. We'll get back to it later.

- `range(n)` generates a sequence of integers starting at 0 (ends before `n`)
  - a sequence of n elements

Simple Repeat Loop

- Run this piece of code:

```
for i in range(5):
   print "Hello!"
```

  - Variable `i` has not been used in the body of the loop.
  - This is just a simple repeat loop
  - The user could choose the number of times to repeat.

```
n = int(raw_input("Enter the number of times to repeat: "))

for i in range(n):
   print "This is repetitious!"
```

Simple Repeat Loop

- We can re-write the repeat loops with `while`:

```
for i in range(5):
   print "Hello!"
```

```
i = 0
while i < 5:
   print "Hello!"
   i += 1
```

Simple Repeat Loop

- We can re-write the repeat loops with `while`:

```
for i in range(5):
   print "Hello!"
```

```
i = 0
while i < 5:
   print "Hello!"
   i += 1
```

- These two programs are doing the same task:

```
n = int(raw_input("Enter the number of times to repeat: "))

for i in range(n):
   print "This is repetitious!"
```

```
n = int(raw_input("Enter the number of times to repeat: "))
i = 0
while i < n:
   print "This is repetitious!"
   i += 1
```

More Examples

- Write a Python code which computes this summation: (you get the value of n from the user)

  1+2+3+4+5+6+… + n

More Examples

- Write a Python code which computes this summation: (you get the value of n from the user)

  1+2+3+4+5+6+… + n

```
num = int(raw_input("Enter an integer number: "))
sum = 0
i = 1

while i <= num:
    sum += i
    i += 1

print "summation:\t", sum
```

More Examples

- Write a Python code which computes this summation: (you get the value of n from the user)

  1+2+3+4+5+6+… + n

```
num = int(raw_input("Enter an integer number: "))
sum = 0

for i in range(num):
    sum += i

print "summation:\t”, sum
```

More Examples

- Write a Python code which computes this summation: (you get the value of n from the user)

  1+2+3+4+5+6+… + n

```
num = int(raw_input("Enter an integer number: "))
sum = 0

for i in range(num):
    sum += i

print "summation:\t", sum
```

Be careful!!

[0, 1, 2, …, num-1]

More Examples

- Write a Python code which computes this summation: (you get the value of n from the user)

  1+2+3+4+5+6+… + n

```
num = int(raw_input("Enter an integer number: "))
sum = 0

for i in range(num+1):
    sum += i

print "summation:\t", sum
```

[0, 1, 2, …, num]

More Examples

- Write a Python code which computes this summation: (you get the value of n from the user)

  1+2+3+4+5+6+… + n

```
num = int(raw_input("Enter an integer number: "))
sum = 0

for i in range(1, num+1):
    sum += i

print "summation:\t", sum
```

[1, 2, …, num]

range(x, y) generates a sequence:

[x, x+1, …, y-1]

More Examples

- Write a Python code which computes this summation: (you get the value of n from the user)

  $1+4+9+16+25+36+\ldots + n^2$

More Examples

- Write a Python code which computes this summation: (you get the value of n from the user)

  $1+4+9+16+25+36+\ldots + n^2$

```
num = int(raw_input("Enter an integer number: "))
sum = 0
i = 1

while i <= num:
    x = i*i
    sum += x
    i += 1

print "summation:\t", sum
```

More Examples

- Write a Python code which computes this summation: (you get the value of n from the user)

  $1+4+9+16+25+36+\ldots + n^2$

```
num = int(raw_input("Enter an integer number: "))
sum = 0

for i in range(1, num+1):
    x = i*i
    sum += x

print "summation:\t", sum
```

More Examples

- Write a Python code which computes this summation: (you get the value of n from the user)

  $1+3+5+7+\ldots +(2n+1)$

More Examples

- Write a Python code which computes this summation: (you get the value of n from the user)

  1+3+5+7+… +(2n+1)

```
num = int(raw_input("Enter an integer number: "))
sum = 0
i = ??

while i <= num:
    x = 2*i + 1
    sum += x
    i += 1

print "summation:\t", sum
```

More Examples

- Write a Python code which computes this summation: (you get the value of n from the user)

  1+3+5+7+… +(2n+1)

```
num = int(raw_input("Enter an integer number: "))
sum = 0
i = 0

while i <= num:
    x = 2*i + 1
    sum += x
    i += 1

print "summation:\t", sum
```

More Examples

- Write a Python code which computes this summation: (you get the value of n from the user)

  1+3+5+7+… +(2n+1)

```
num = int(raw_input("Enter an integer number: "))
sum = 0

for i in range(num+1):
    x = 2*i + 1
    sum += x

print "summation:\t", sum
```

More Examples

- Write a Python code to compute this summation: (you get the value of n from the user)

  1/2 + 1/3+ 1/4 + 1/5 + 1/7+… + 1/n

More Examples

- Write a Python code to compute this summation: (you get the value of n from the user)

  1/2 + 1/3+ 1/4 + 1/5 + 1/7+… + 1/n

```
num = int(raw_input("Enter an integer number: "))
sum = 0

for i in range(2, num+1):
    x = 1.0/i
    sum += x

print "summation:\t", sum
```

range(a, b): generates a sequence [a, a+1, …, b-1]

More Examples

- Write a Python code which computes this summation: (you get the value of n from the user)

  1-3+5-7+… ± (2n+1)

  Note: (+ or - depends on value of n on the last term)

More Examples

- Write a Python code which computes this summation: (you get the value of n from the user)

  1-3+5-7+… ± (2n+1)

  Note: (+ or - depends on value of n on the last term)

```
num = int(raw_input("Enter an integer number: "))
sign = 1
sum = 0

for i in range(num+1):
   x = sign*(2*i+1)
   sum += x
   sign *= -1

print "summation:\t", sum
```

More Examples

- Write a Python code which computes this summation:

  1+4+7+10+13+16+… +37+40

More Examples

- Write a Python code which computes this summation:

  1+4+7+10+13+16+… +37+40

```
number = 1
sum = 0

while number <= 40:
    sum += number
    number += 3

print "summation:\t”, sum
```

Example: Print Patterns

- Write a Python program to take a number and print patterns like: (the following examples are patterns of size 5)
  - **We have not learned about strings yet, so you cannot use string methods (even if you know them)**

```
First pattern!
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
```

```
Second pattern!
*
* *
* * *
* * * *
* * * * *
```

```
Third pattern!
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
```

```
Forth pattern!
        1
      2 2 2
    3 3 3 3 3
  4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5
```

```
Fifth pattern!
        1
      2 1 2
    3 2 1 2 3
  4 3 2 1 2 3 4
5 4 3 2 1 2 3 4 5
```

Example: Print Patterns

```
First pattern!
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
```

```
num = int(raw_input("enter a number: “))
i = 0
while i < num:
    print "* * * * * * * * * *”
    i += 1
```

Example: Print Patterns

```
* * * * * * * * * *
```

```
j = 0
while j < 10:
   print "*" ,
   j += 1
print
```

Example: Print Patterns

```
* * * * * * * * * *
```

```
j = 0
while j < 10:
    print "*" ,
    j += 1
print
```

will not print newline character and print the next value in the same line

Just print a newline (it is the end of the line)

Example: Print Patterns

```
First pattern!
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
```

```
num = int(raw_input("enter a number: "))

i = 0
while i < num:
    j = 0
    while j < 10:
        print "*" ,
        j += 1

    print
    i += 1
```

Example: Print Patterns

```
Second pattern!
*
* *
* * *
* * * *
* * * * *
```

Example: Print Patterns

```
Second pattern!
*
* *
* * *
* * * *
* * * * *
```

```
num = int(raw_input("enter a number: "))

i = 0
while i < num:
    j = 0
    while j < i:
        print "*" ,
        j += 1

    print
    i += 1
```

Example: Print Patterns

```
Third pattern!
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
```

Example: Print Patterns

```
Third pattern!
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
```

```
num = int(raw_input("enter a number: "))

i = 0
while i < num:
    j = 0
    while j < 2*i+1:
        print "*" ,
        j += 1

    print
    i += 1
```

Example: Print Patterns

```
num = int(raw_input("enter a number: "))

i = 0
while i < num:
    j = 0
    while j < 2*i+1:
        print "*" ,
        j += 1

    print
    i += 1
```

```
Third pattern!
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
```

```
Output
*
* * *
* * * * *
* * * * * * *
* * * * * * * * *
```

Example: Print Patterns

```
num = int(raw_input("enter a number: "))

i = 0
while i < num:
    j = 0
    while j < num-i-1:
        print " " ,
        j += 1

    j = 0
    while j < 2*i+1:
        print "*" ,
        j += 1

    print
    i += 1
```

```
Third pattern!
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
```

Example: Print Patterns

```
Forth pattern!
        1
      2 2 2
    3 3 3 3 3
  4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5
```

Example: Print Patterns

```
num = int(raw_input("enter a number: "))

i = 0
while i < num:
    j = 0
    while j < num-i-1:
        print " " ,
        j += 1

    j = 0
    while j < 2*i+1:
        print i+1 ,
        j += 1

    print
    i += 1
```

```
Forth pattern!
        1
      2 2 2
    3 3 3 3 3
  4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5
```

Example: Print Patterns

```
num = int(raw_input("enter a number: "))

i = 0
while i < num:
    j = 0
    while j < num-i-1:
        print " " ,
        j += 1

    j = 0
    while j < 2*i+1:
        print (i+1)%10 ,
        j += 1

    print
    i += 1
```

```
Forth pattern!
        1
      2 2 2
    3 3 3 3 3
  4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5
```

Example: Print Patterns

```
Fifth pattern!
        1
      2 1 2
    3 2 1 2 3
  4 3 2 1 2 3 4
5 4 3 2 1 2 3 4 5
```

Example: Print Patterns

```
Fifth pattern!
        1
      2 1 2
    3 2 1 2 3
  4 3 2 1 2 3 4
5 4 3 2 1 2 3 4 5
```

```
num = int(raw_input("enter a number: "))

i = 0
while i < num:
    j = 0
    while j < num-i-1:
        print " " ,
        j += 1

    j = 0
    while j < 2*i+1:
        print (i+1)%10 ,
        j += 1

    print
    i += 1
```

Is one loop enough?!

Example: Print Patterns

```
Fifth pattern!
        1
      2 1 2
    3 2 1 2 3
  4 3 2 1 2 3 4
5 4 3 2 1 2 3 4 5
```

```
num = int(raw_input("enter a number: "))

i = 0
while i < num:
    j = 0
    while j < num-i-1:
        print " " ,
        j += 1

    j = i+1
    while j > 0:
        print j%10 ,
        j -= 1

    print
    i += 1
```

prints a sequence:

i+1 i i-1 … 1

Example: Print Patterns

```
num = int(raw_input("enter a number: "))

i = 0
while i < num:
    j = 0
    while j < num-i-1:
        print " " ,
        j += 1

    j = i+1
    while j > 0:
        print j%10 ,
        j -= 1

    print
    i += 1
```

```
Fifth pattern!
        1
      2 1 2
    3 2 1 2 3
  4 3 2 1 2 3 4
5 4 3 2 1 2 3 4 5
```

prints a sequence:

i+1 i i-1 … 1

```
Output
        1
      2 1
    3 2 1
  4 3 2 1
5 4 3 2 1
```

Example: Print Patterns

```
Fifth pattern!
          1
        2 1 2
      3 2 1 2 3
    4 3 2 1 2 3 4
  5 4 3 2 1 2 3 4 5
```

```
num = int(raw_input("enter a number: "))

i = 0
while i < num:
    j = 0
    while j < num-i-1:
        print " " ,
        j += 1

    j = i+1
    while j > 0:
        print j%10 ,
        j -= 1

    j = 2
    while j <= i+1:
        print j%10 ,
        j += 1

    print
    i += 1
```

prints a sequence:

i+1 i i-1 … 1

prints the second part:

2 3 … i i+1

More examples about `for` loop working on lists

`for` Example

- Suppose I have a list of values called `items`, and I want to print out each item and number them successively.
  - for example if `items` is `['red', 'orange', 'yellow', 'green']` the output should look like:

```
1 red
2 orange
3 yellow
4 green
```

`for` Example

- Suppose I have a list of values called `items`, and I want to print out each item and number them successively.
  - for example if `items` is `['red', 'orange', 'yellow', 'green']` the output should look like:

```
1 red
2 orange
3 yellow
4 green
```

```
items = ["red","orange","yellow","green"]

for x in items:
   print x
```

Wrong?! it just print the items

`for` Example

- Suppose I have a list of values called `items`, and I want to print out each item and number them successively.
  - for example if `items` is `['red', 'orange', 'yellow', 'green']` the output should look like:

```
1 red
2 orange
3 yellow
4 green
```

```
items = ["red","orange","yellow","green"]
i = 1
for x in items:
   print i, x
   i += 1
```

`for` Example

- Suppose I have a list of integer numbers, called `numbers`, and I want to print out even numbers and their corresponding position in the list.

```
numbers = [3,42,90,1,5,8,50]
```

```
2 42
3 90
6 8
7 50
```

`for` Example

- Suppose I have a list of integer numbers, called `numbers`, and I want to print out even numbers and their corresponding position in the list.

```
numbers = [3,42,90,1,5,8,50]
i = 1

for num in numbers:
    if num % 2 == 0:
        print i, num
    i += 1
```

```
2 42
3 90
6 8
7 50
```

Example (find min/max)

- Suppose I have a list of values, and I want to find the largest value. (assume that values are positive)

Example (find min/max)

- Suppose I have a list of values, and I want to find the largest value. (assume that values are positive)

```
items = [3,45,1,90,5,8,51]
max = -1

for x in items:
   if x > max:
       max = x

print "maximum value:", max
```

```
maximum value: 90
```

Example (find min/max)

- Suppose I have a list of values, and I want to find the largest value. (assume that values are positive)

```
items = [3,45,1,90,5,8,51]
max = -1

for x in items:
   if x > max:
       max = x

print "maximum value:", max
```

`max` is initialized with the a value less than the smallest possible value

```
maximum value: 90
```

Example (find min/max)

- Suppose I have a list of values, and I want to find the largest value (assume that values are positive)

```
items = [3,45,1,90,5,8,51]
max = -1

for x in items:
   if x > max:
       max = x

print "maximum value:", max
```

for each value in `items`, we compare it to `max`, if it is greater, we replace it

```
maximum value: 90
```

Example (find min/max)

- Suppose I have a list of values, and I want to find the largest value (assume that values are positive)
  - modify this program so it also prints the position (or index) of the largest value in the list or sequence (**starting from 0**)

```
items = [3,45,1,90,5,8,51]
max = -1

for x in items:
   if x > max:
       max = x

print "maximum value:", max
```

```
maximum value: 90
at index: 3
```

Example (find min/max)

- Suppose I have a list of values, and I want to find the largest value. (assume that values are positive)
  - modify this program so it also prints the position (or index) of the largest value in the list or sequence (**starting from 0**)

```
items = [3,45,1,90,5,8,51]
max = -1
i = 0
for x in items:
   if x > max:
       max = x
   i += 1

print "maximum value:", max
print "at index:", i
```

Example (find min/max)

- Suppose I have a list of values, and I want to find the largest value. (assume that values are positive)
  - modify this program so it also prints the position (or index) of the largest value in the list or sequence (**starting from 0**)

```
items = [3,45,1,90,5,8,51]
max = -1
i = 0
for x in items:
   if x > max:
       max = x
   i += 1

print "maximum value:", max
print "at index:", i
```

what's wrong?

```
maximum value: 90
at index: 7
```

Example (find min/max)

- Suppose I have a list of values, and I want to find the largest value. (assume that values are positive)
  - modify this program so it also prints the position (or index) of the largest value in the list or sequence (**starting from 0**)

```
items = [3,45,1,90,5,8,51]
max = -1
i = 0
for x in items:
   if x > max:
       max = x
   i += 1

print "maximum value:", max
print "at index:", i
```

we need another variable to keep the correct index

```
maximum value: 90
at index: 3
```

Example (find min/max)

- Suppose I have a list of values, and I want to find the largest value. (assume that values are positive)
  - modify this program so it also prints the position (or index) of the largest value in the list or sequence (**starting from 0**)

```
items = [3,45,1,90,5,8,51]
max = -1
index = -1
i = 0
for x in items:
   if x > max:
       max = x
       index = i
   i += 1

print "maximum value:", max
print "at index:", index
```

when whenever `max` is updated, `index` should be updated to the current value of `i`

```
maximum value: 90
at index: 3
```

Example (find min/max)

- Suppose I have a list of values, and I want to find the ***smallest*** value.
  - modify this program so it also prints the position (or index) of the smallest value in the list or sequence (**starting from 0**)

```
items = [3,45,1,90,5,8,51]
max = -1
index = -1
i = 0
for x in items:
   if x > max:
       max = x
       index = i
   i += 1

print "maximum value:", max
print "at index:", index
```

```
minimum value: 1
at index: 2
```

Example (find min/max)

- Suppose I have a list of values, and I want to find the ***smallest*** value.
  - modify this program so it also prints the position (or index) of the smallest value in the list or sequence (**starting from 0**)

```
items = [3,45,1,90,5,8,51]
min = 100000000
index = -1
i = 0
for x in items:
   if x < min:
       min = x
       index = i
   i += 1

print "minimum value:", min
print "at index:", index
```

`min` should be initialized with the a very large value (larger than the largest possible value in our list)

```
minimum value: 1
at index: 2
```

Constant Value `None`

- `None` is a special constant value which we can store in a variable to mark the variable as "***empty***"
  - Before the loop starts, `min` value is `None` (since we have not seen any value from the list)

```
items = [3,45,1,90,5,8,51]
min = None
index = -1
i = 0

for x in items:
    if min is None or x < min:
        min = x
        index = i
    i += 1

print "minimum value:", min
print "at index:", index
```

Just in the first iteration `min` is `None`

```
minimum value: 1
at index: 2
```

- Write a program to compute the summation of some positive numbers entered by the users (you should ask the user to enter the number and continue as long as the user wish). If the user enters a negative number you should print a message for the user that the input is invalid and ignore that input and continue

Recap from last lecture!

```
sum = 0
while True:
    x = raw_input("Enter an integer: ")
    x = int(x)
    if x < 0:
        print "invalid input! Please try again!"
        continue

    sum += x

    answer = raw_input("Do you want to continue? (y/n) ")
    if answer == 'n':
        break

print "sum:", sum
```

```
Enter an integer: 2
Do you want to continue? (y/n) y
Enter an integer: -4
invalid input! Please try again!
Enter an integer: 10
Do you want to continue? (y/n) n
sum: 12
```

- update the previous program to also find the maximum value (in addition to summation of numbers)

```
sum = 0
max = None

while True:
    x = raw_input("Enter an integer: ")
    x = int(x)
    if x < 0:
        print "invalid input! Please try again!"
        continue

    sum += x
    if max is None or x > max:
        max = x


    answer = raw_input("Do you want to continue? (y/n) ")
    if answer == 'n':
        break

print "sum:", sum
print "max:", max
```

```
Enter an integer: 2
Do you want to continue? (y/n) y
Enter an integer: -4
invalid input! Please try again!
Enter an integer: 10
Do you want to continue? (y/n) n
sum: 12
max: 10
```

Suggested Reading

- Chapter 5

More Examples

- Write a code to check if a number is prime
  - prime numbers: numbers that are only divisible by themselves and 1

More Examples

- Write a code to check if a number is prime
  - prime numbers: numbers that are only divisible by themselves and 1

```
num = int(raw_input("Enter an integer number: "))
if num <= 1:
   flag = False
else:
   flag = True

i = 2
while i < num:
    if num % i == 0:
        flag = False
        break
    i += 1

if flag:
    print num,"is prime"
else:
    print num,"is not prime"
```

More Examples

- Write a code to check if a number is prime
  - prime numbers: numbers that are only divisible by themselves and 1

```
num = int(raw_input("Enter an integer number: "))
if num <= 1:
   flag = False
else:
   flag = True

i = 2
while i < num:
    if num % i == 0:
        flag = False
        break
    i += 1

if flag:
    print num,"is prime"
else:
    print num,"is not prime"
```

Practice: can you decrease the number of iterations and make your program faster?