SAMS
Programming A/B

Lecture #3 – Loops
July 10, 2017

Mark Stehlik

Outline for Today

- Iteration
  - For loops
  - While loops

Why Iteration?

- More generality so more power
- Example: remember the tip function:

```
def tip(total):
  return total * .18

>>> tip(25)
4.5
```

But what if we want a table of tip amounts?

Getting a table of results (the hard way)

```
def tip_table_1():
  print(tip(10))
  print(tip(11))
  print(tip(12))
  print(tip(13))
  # etc. for more values

>>> tip_table_1()
1.7999999999999998
1.98
2.16
2.34
```

Getting a table of results (the easy way)

```
def tip_table_2(low, high):
  for amount in range(low, high+1):
    print(tip(amount))

>>> tip_table_2(10,20)
1.7999999999999998
1.98
2.16
2.34
2.52
2.6999999999999997
2.88
3.06
3.2399999999999998
3.42
3.5999999999999996
>>>
```

for Loop (simple version)

```
for loop_variable in range(n):
     loop body
```

- The loop variable is a new variable name
- The loop body is one or more instructions that you want to repeat.
- If n > 0, the for loop repeats the loop body n times.
- If n <= 0, the entire loop is skipped.
- Remember to indent loop body

for Loop Example

Loop variable

```
for i in range(5):
    print("hello world")
```

```
hello world
hello world
hello world
hello world
hello world
```

for Loops and range()

- for loop
  - Used to iterate over a known interval/set of values
  - range() is your friend! (but ints only, if you please!)
- range(), a Python built-in, has lots of options:
  - range(n) – generates the numbers 0 to n-1
  - range(start, end) – generates start to end-1
  - range(start, end, increment) – generates start to end-1 by increment

Some range examples

-
  ```
  for num in range(10):
          print(num)  #prints ?
  ```

-
  ```
  for num in range(5,11):
          print(num)  #prints ?
  ```

-
  ```
  for num in range(5, 11, 2):
          print(num)  #prints ?
  ```

-
  ```
  for num in range(15, 5, -2):
          print(num)  #prints ?
  ```

Detour: some printing options

```
>>> for i in range(5):
...      print(i, end=" ")
0 1 2 3 4 >>>
>>>
>>> for i in range(5):
>>>      print(i, end="")
01234>>>
```

Blank space after printing expression

No space after printing expression

The default is `end="\n”`.

That is, when you don’t include the `end` argument

`print` will go to the next line after printing the expression.

Accumulating an answer

```
def total():
  # sums first 5 positive integers
  sum = 0 # initialize accumulator
  for i in range(1, 6):
     sum = sum + i # update accumulator
  return sum # return accumulated result

>>> sum()
15
```

Generalizing sum

```
def total(n):
  # sums the first n positive integers
  sum = 0 # initialize
  for x in range(n + 1):
     sum = sum + x # update
  return sum # accumulated result
```

```
sum(6)        returns 21
sum(100)      returns 5050
sum(15110)    returns 114163605
```

Danger! Don’t change the loop variable!

```
for i in range(5):
  print(i, end=" ")
  i = 10

0 1 2 3 4
```

Even if you modify the loop variable in the loop, it will be reset to its next expected value in the next iteration.

```
for i in range(5):
  i = 10
  print(i, end=" ")

10 10 10 10 10
```

NEVER modify the loop variable inside a `for` loop.

Nested for Loop example

What does the following nested loop do?

```
for row in range(1,11):
        for col in range(1,11):
                print(row * col, " ", end="")
        print()
print()
```

While loop

- An *indefinite* loop – used when you don’t know the exact interval
- while <condition>:

  ```
      statement(s)
  ```
- As long as the <condition> is true, the loop will execute

While loop example

```
sum = 0
count = 0
num = int(input("Enter a number: "))
while num != -1:    # -1 is a sentinel value
        sum = sum + num
        count = count + 1
        num = int(input("Enter a number: "))
print("average is", sum/count)
```

More coding examples…