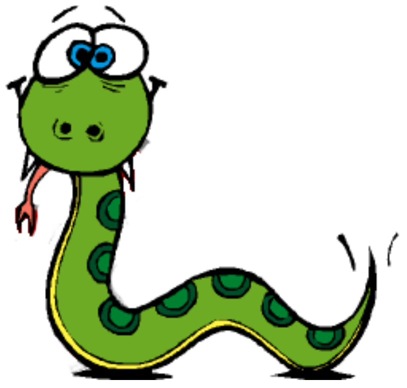
IDCE 302: Chapter 6

Iterations

Jie Tian, PhD

Clark University



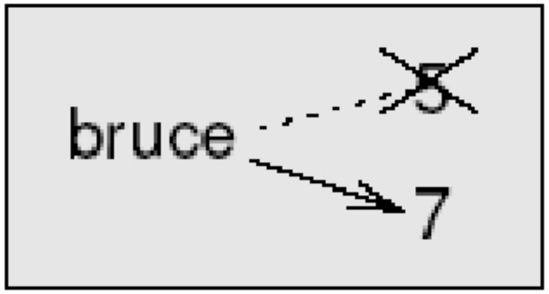
Outline

- Multiple Assignment & Value Swapping
- The while loop
- Encapsulation & Generalization
- Revisit of local variables

Multiple Assignment

- It is allowed to assign a value to a variable multiple times

```
>>> bruce = 5
>>> print bruce
5
>>> bruce = 7
>>> print bruce
7
>>> x = 0
>>> x = x + 1
print x
1
```



Swap Values

```
>>> intA = -128
>>> intB = 127
```

Remember this!

```
>>> tmp = intA
>>> intA = intB
>>> intB = tmp

>>> print intB
-128
>>> print intA
127
```

The `while` statement

- Computers are often used to **automate repetitive tasks**.
- Python provides statements (**for** and **while**) to achieve iteration.

Step 1
- Evaluate the condition, yielding **True** or **False**

Step 2
- If the condition is **false**, exit the statement and continue execution at the next statement.

Step 3
- If the condition is **true**, execute the body and then go back to step 1.

while example

```
>>> n=5
>>> while n>0:
...     print n
...     n=n-1
...
5
4
3
2
1
```

Revisit the countdown example

```
def countdown(n):
    if n == 0: #base case
        print "Blastoff!"
    else: #general  case
        print n
        countdown(n-1)
```

```
def  countdown(n):
    while  n  > 0:
        print n
        n = n-1
    print  “Blastoff!"
```

```
# test case
countdown(5)
5
4
3
2
1
Blastoff!
```

`while` loop example 1: get a sum

- Task is to get the sum of 1, 2, …, and 99

```
>>> n=1
>>> sum=0
>>> while n<=99:
...     sum = sum + n
...     n = n+1
...
>>> print sum
4950
```

while loop example 2: selective sum

- Task is to add all the even numbers between 1 and 99

```
n=1
sum=0
while n<=99:
    if n % 2 == 0:
        sum = sum + n
    n = n+1

print sum
```

Infinite While Loop

```
n=1
sum=0
while n<=99:
    if n % 2 == 0:
        sum = sum + n
    n = n+1 # what if we comment out this line??
print sum
```

Break

- Sometimes you want to end a loop when half way through the body.
- break is useful in such cases.

```
while True:
    line = raw_input(‘Type Something: ')
    if line == 'done':
        break
    print line

print 'Done!'
```

while loop example 3

x	log2 x
1.0	0.0
2.0	1.0
3.0	1.58496250072
4.0	2.0
5.0	2.32192809489
6.0	2.58496250072
7.0	2.80735492206
8.0	3.0
9.0	3.16992500144

$$\log_2 x = \frac{\ln(x)}{\ln(2)}$$

Print a look-up table of $\log_2 X$

```python
"""
input:
   upperbound: an upperbound of the table
output:
    a two-column table:
            value1     2-based log value1
            value2     2-based log value2
            ...
"""
import math

def log2Table(upperbound):
    x = 1.0
    while x <=upperbound:
        print x, '\t', math.log(x)/math.log(2.0)
        x = x+1  # x * 2.0
# test
log2Table(9)
```

while example 4

```
1       2       3       4       5       6       7       8       9
2       4       6       8       10      12      14      16      18
3       6       9       12      15      18      21      24      27
4       8       12      16      20      24      28      32      36
5       10      15      20      25      30      35      40      45
6       12      18      24      30      36      42      48      54
7       14      21      28      35      42      49      56      63
8       16      24      32      40      48      56      64      72
9       18      27      36      45      54      63      72      81
```

1*(1	2	3	4	5	6	7	8	9)
2*(1	2	3	4	5	6	7	8	9)
3*(1	2	3	4	5	6	7	8	9)
4*(1	2	3	4	5	6	7	8	9)
5*(1	2	3	4	5	6	7	8	9)
6*(1	2	3	4	5	6	7	8	9)
7*(1	2	3	4	5	6	7	8	9)
8*(1	2	3	4	5	6	7	8	9)
9*(1	2	3	4	5	6	7	8	9)

```
"""
  Just create the 1st row
"""

i=1

j=1
while j<=9:
    print i*j, '\t', # there is a comma
        # continue to print in the same line
    j = j+1
```

```
"""
  Just create the 2nd row
"""

i=2

j=1
while j<=9:
    print i*j, '\t', # there is a comma!
    j = j+1
```

```
"""
    I don’t want to write 9 of them! No…..
"""

i=1
while i<=9:
    j = 1
    while j<=9:
        # output formatting may be used here
        print i*j, ‘\t', # there is a comma!
        j = j+1
    print ‘\n’ # new line
    i = i+1
```

```
"""
    Create a generalized function for making 2D tables
"""

def multipTable(iValRow, iValCol):
    i=1
    while i<=iValRow:
        j=1
        while j<=iValCol:

            print i*j, '\t', # there is a coma!
            j  = j+1
        print ‘\n’
        i=  i+1

#  test
multipTable(9,9)
```

Encapsulation & Generalization

Wrapping a piece of code up in a function is called **encapsulation**.

```
def multipTable(iRow, iCol):
    i=1;
    while i<=iRow:
        printMultiples(i, iCol)     # i is the multiplier
        print '\n'
        i= i+1

def printMultiples(iMult, iCol):
    i = 1
    while i <= iCol:
        print iMult*i, '\t',
        i = i + 1

#   test
multipTable(9,9)
```

Benefits of Using Functions

- Partition complex problems into simple ones, solve one problem a time (divide and conquer).
- Make programs easy to debug.
- Functions can be reused.

Summary

- Variable assignment (multiple assignment, swapping)
- while loop (finite, infinite, break)
- Local variables (not recognized only within the function)
- Using functions is good! Encapsulation!

Exercise

Use a while loop to add all numbers from 1 to 20 that can be divided by 5.

```
#  Hint: below is the code  for adding  even
numbers between 1 to 99

# Add all even number 1, 2, …, 99

n=1
sum=0
while n<=99:
    if  n % 2  == 0:
        sum = sum + n
    n = n+1

print sum
```

Write a script to print the table below.

1	2	3	4	5
1	4	9	16	25
1	8	24	64	125