DM536
Programming A

Peter Schneider-Kamp

petersk@imada.sdu.dk

http://imada.sdu.dk/~petersk/DM536/

ITERATION

Multiple Assignment Revisited

- as seen before, variables can be assigned multiple times
- assignment is NOT the same as equality
- it is not symmetric, and changes with time

- Example:

  a = 42

  …

  b = a

  …

  a = 23

from here, a and b are equal

from here, a and b are different

Updating Variables

- most common form of multiple assignment is *updating*
- a variable is assigned to an expression containing that variable

- Example:

```
x = 23
for i in range(19):
    x = x + 1
```

- adding one is called *incrementing*

- expression evaluated BEFORE assignment takes place
- thus, variable needs to have been *initialized* earlier!

Iterating with While Loops

- iteration = repetition of code blocks
- can be implemented using recursion (countdown, polyline)

- while statement:

  <while-loop> => while <cond>:

  <instr$_1$>; <instr$_2$>; <instr$_3$>

- Example:

```
def countdown(n):
    while n > 0:
        print n, "seconds left!"
        n = n - 1
    print "Ka-Boom!"
countdown(3)
```

n == 0

False

Termination

- Termination = the condition is eventually False
- loop in countdown obviously terminates:

```
while n > 0:      n = n - 1
```

- difficult for other loops:

```
def collatz(n):
    while n != 1:
        print n,
        if n % 2 == 0:          # n is even
            n = n / 2
        else:                   # n is odd
            n = 3 * n + 1
```

Termination

- Termination = the condition is eventually False
- loop in countdown obviously terminates:

```
while n > 0:        n = n - 1
```

- can also be difficult for recursion:

```
def collatz(n):
    if n != 1:
        print n,
        if n % 2 == 0:          # n is even
            collatz(n / 2)
        else:                   # n is odd
            collatz(3 * n + 1)
```

Breaking a Loop

- sometimes you want to *force* termination
- Example:

```
while True:
    num = raw_input('enter a number (or "exit"):\n')
    if num == "exit":
        break
    n = int(num)
    print "Square of", n, "is:", n**2
print "Thanks a lot!"
```

Approximating Square Roots

- Newton's method for finding root of a function f:
  1. start with some value x_0
  2. refine this value using $x_{n+1} = x_n - f(x_n) / f'(x_n)$
- for square root of a: $f(x) = x^2 - a$ $f'(x) = 2x$
- simplifying for this special case: $x_{n+1} = (x_n + a / x_n) / 2$

- Example 1:

```
while True:
    print xn
    xnp1 = (xn + a / xn) / 2
    if xnp1 == xn:
        break
    xn = xnp1
```

Approximating Square Roots

- Newton's method for finding root of a function f:
  1. start with some value x_0
  2. refine this value using $x_{n+1} = x_n - f(x_n) / f'(x_n)$

- Example 2:

```
def f(x):      return x**3 - math.cos(x)
def f1(x):     return 3*x**2 + math.sin(x)
while True:
    print xn
    xnp1 = xn - f(xn) / f1(xn)
    if xnp1 == xn:
        break
    xn = xnp1
```

Approximating Square Roots

- Newton's method for finding root of a function f:
  1. start with some value x_0
  2. refine this value using $x_{n+1} = x_n - f(x_n) / f'(x_n)$

- Example 2:

```
def f(x):      return x**3 - math.cos(x)
def f1(x):     return 3*x**2 + math.sin(x)
while True:
    print xn
    xnp1 = xn - f(xn) / f1(xn)
    if math.abs(xnp1 - xn) < epsilon:
        break
    xn = xnp1
```

Algorithms

- algorithm = mechanical problem-solving process
- usually given as a step-by-step procedure for computation

- Newton's method is an example of an algorithm
- other examples:
  - addition with carrying
  - subtraction with borrowing
  - long multiplication
  - long division

- directly using Pythagora's formula is not an algorithm

Divide et Impera

- latin, means "divide and conquer" (courtesy of Julius Caesar)
- **Idea:** break down a problem and recursively work on parts
- Example: guessing a number by bisection

```
def guess(low, high):
    if low == high:
        print "Got you! You thought of: ", low
    else:
        mid = (low+high) / 2
        ans = raw_input("Is "+str(mid)+" correct (>, =, <)?")
        if ans == ">":      guess(mid,high)
        elif ans == "<":    guess(low,mid)
        else:               print "Yeehah! Got you!"
```

Debugging Larger Programs

- assume you have large function computing wrong return value
- going step-by-step very time consuming

- **Idea:** use bisection, i.e., half the search space in each step

1. insert intermediate output (e.g. using print) at mid-point
2. if intermediate output is correct, apply recursively to 2nd part
3. if intermediate output is wrong, apply recursively to 1st part

STRINGS

Strings as Sequences

- strings can be viewed as 0-indexed sequences
- Examples:

  "Slartibartfast"[0] == "S"

  "Slartibartfast"[1] == "l"

  "Slartibartfast"[2] == "Slartibartfast"[7]

  "Phartiphukborlz"[-1] == "z"

- grammar rule for expressions:

  <expr> => … | <expr$_1$>[<expr$_2$>]

- <expr$_1$> = expression with value of type string
- index <expr$_2$> = expression with value of type integer
- negative index counting from the back

Length of Strings

- length of a string computed by built-in function len(object)

- Example:

```
name = "Slartibartfast"
length = len(name)
print name[length-4]
```

- Note: name[length] gives runtime error

- identical to write name[len(name)-1] and name[-1]
- more general, name[len(name)-a] identical to name[-a]

Traversing with While Loop

- many operations go through string one character at a time
- this can be accomplished using
  - a while loop,
  - an integer variable, and
  - index access to the string
- Example:

```
index = 0
while index < len(name):
    letter = name[index]
    print letter
    index = index + 1
```

Traversing with For Loop

- many operations go through string one character at a time
- this can be accomplished *easier* using
  - a for loop and
  - a string variable

- Example:

```
for letter in name:
    print letter
```

Generating Duck Names

- What does the following code do?

```
prefix = "R"
infixes = "iau"
suffix = "p"
for infix in infixes:
    print prefix + infix + suffix
```

- … and greetings from Andebyen!

String Slices

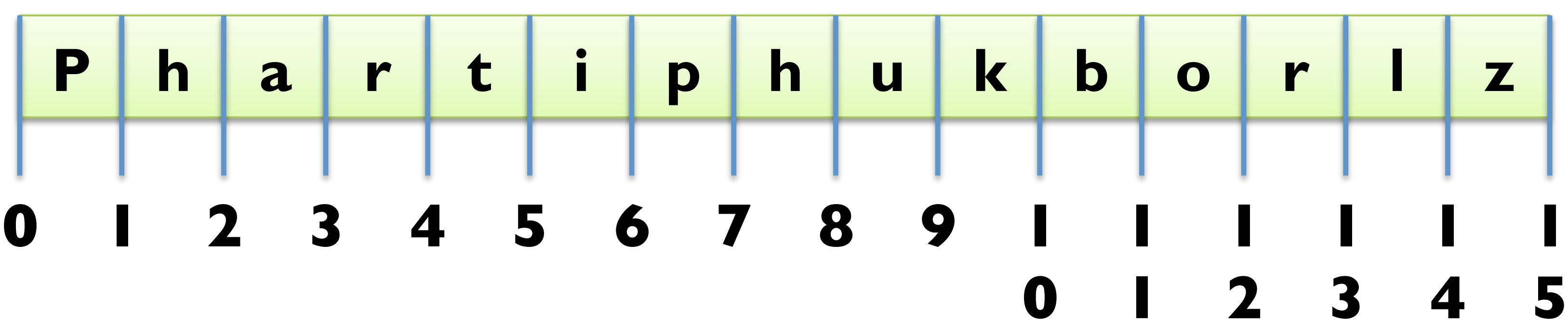
- slice = part of a string
- Example 1:

```
name = "Phartiphukborlz"
print name[6:10]
```

- one can use negative indices:

```
name[6:-5] == name[6:len(name)-5]
```

- view string with indices before letters:

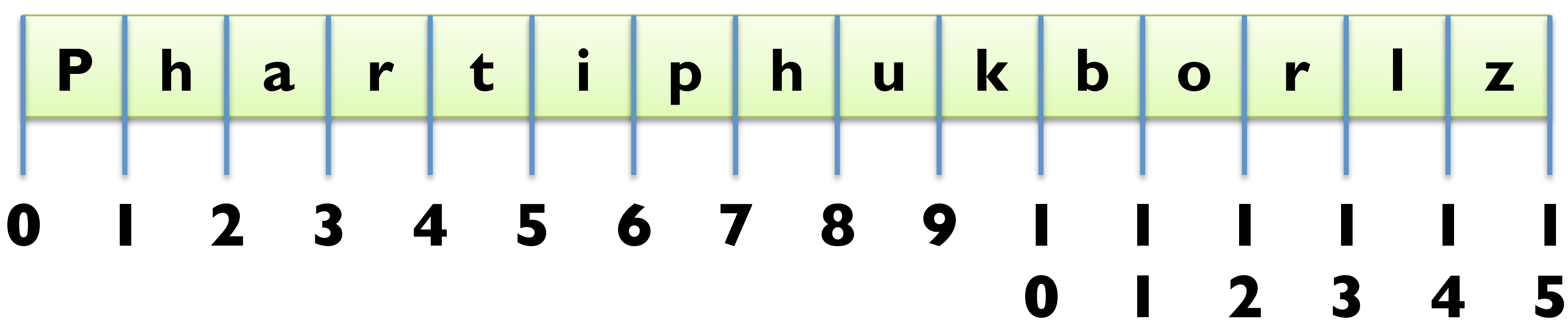


String Slices

- slice = part of a string
- Example 2:

```
name = "Phartiphukborlz"
print name[6:6]        # empty string has length 0
print name[:6]         # no left index = 0
print name[6:]         # no right index = len(name)
print name[:]          # guess ;)
```

- view string with indices before letters:



Changing Strings

- indices and slices are read-only (*immutable*)
- you cannot assign to an index or a slice:

```
name = "Slartibartfast"
name[0] = "s"
```

- change strings by building new ones
- Example 1:

```
name = "Slartibartfast"
name = "s" + name[1:]
```

- Example 2:

```
name = "Anders And"
name2 = name[:6] + "ine" + name[6:]
```

Searching in Strings

- indexing goes from index to letter
- reverse operation is called find (*search*)
- Implementation:

```
def find(word, letter):
    index = 0
    while index < len(word):
        if word[index] == letter:
            return index
        index = index + 1
    return -1
```

- Why not use a for loop?

Looping and Counting

- want to count number of a certain letter in a word
- for this, we use a *counter* variable

- Implementation:

```
def count(word, letter):
    count = 0
    for x in word:
        if x == letter:
            count = count + 1
    return count
```

- Can we use a while loop here?

String Methods

- methods = functions associated to a data structure
- calling a method is called *method invocation*
- dir(object): get list of all methods of a data structure
- Example:

```
name = "Slartibartfast"
print name.lower()
print name.upper()
print name.find("a")
print name.count("a")
for method in dir(name):
    print method
help(name.upper)
```

Using the Inclusion Operator

- how to find out if string contained in another string?
- **Idea:** use a while loop and slices

```
def contained_in(word1, word2):
    index = 0
    while index+len(word1) <= len(word2):
        if word2[index:index+len(word1)] == word1:
            return True
        index = index+1
    return False
```

- Python has pre-defined operator in:

```
print "phuk" in "Phartiphukborlz"
```

Comparing Strings

- string comparison is from left-to-right (*lexicographic*)
- Example 1:

  "slartibartfast" > "phartiphukborlz"

- Example 2:

  "Slartibartfast" < "phartiphukborlz"

- **Note:** string comparison is case-sensitive
- to avoid problems with case, use lower() or upper()
- Example 3:

  "Slartibartfast".upper() > "phartiphukborlz".upper()

Debugging String Algorithms

- beginning and end critical, when iterating through sequences
- number of iterations often off by one (*obi-wan error*)
- Example:

```
def is_reverse(word1, word2):
    if len(word1) != len(word2):        return False
    i = 0
    j = len(word2)
    while j > 0:
        if word1[i] != word2[j]:        return False
        i = i + 1;  j = j - 1
    return True
```

Debugging String Algorithms

- beginning and end critical, when iterating through sequences
- number of iterations often off by one (*obi-wan error*)
- Example:

```
def is_reverse(word1, word2):
    if len(word1) != len(word2):        return False
    i = 0
    j = len(word2) - 1
    while j > 0:
        if word1[i] != word2[j]:        return False
        i = i + 1;  j = j - 1
    return True
```

Debugging String Algorithms

- beginning and end critical, when iterating through sequences
- number of iterations often off by one (*obi-wan error*)
- Example:

```
def is_reverse(word1, word2):
    if len(word1) != len(word2):        return False
    i = 0
    j = len(word2) - 1
    while j >= 0:
        if word1[i] != word2[j]:        return False
        i = i + 1;  j = j - 1
    return True
```

Debugging String Algorithms

- beginning and end critical, when iterating through sequences
- number of iterations often off by one (*obi-wan error*)
- Example:

```
def is_reverse(word1, word2):
    if len(word1) != len(word2):        return False
    i = 0
    j = len(word2)
    while j > 0:
        if word1[i] != word2[j-1]:          return False
        i = i + 1;  j = j - 1
    return True
```

HANDLING TEXT FILES

Reading Files

- open files for reading using the open(name) built-in function
  - Example: `f = open("anna_karenina.txt")`
- return value is file object in reading mode (mode 'r')
- we can read all content into string using the read() method
  - Example:

```
content = f.read()
print content[:60]
print content[3000:3137]
```

- contains line endings (here "\r\n")

Reading Lines from a File

- instead of reading all content, we can use method readline()
  - Example:

```
print f.readline()
next = f.readline().strip()
print next
```

- the method strip() removes all leading and trailing whitespace
- whitespace = \n, \r, or \t (new line, carriage return, tab)

- we can also iterate through all lines using a for loop
  - Example:

```
for line in f:
    line = line.strip()
    print line
```

Reading Words from a File

- often a line consists of many words
- no direct support to read words
- string method split() can be used with for loop
  - Example:

```
def print_all_words(f):
    for line in f:
        for word in line.split():
            print word
```

- variant split(sep) using sep instead of whitespace
  - Example:

```
for part in "Slartibartfast".split("a"):
    print part
```

Analyzing Words

- Example 1: words beginning with capital letter ending in "a"

```
def cap_end_a(word):
    return word[0].upper() == word[0]
```

Analyzing Words

- Example 1: words beginning with capital letter ending in "a"

```
def cap_end_a(word):
    return word[0].upper() == word[0] and word[-1] == "a"
```

Analyzing Words

- Example 1: words beginning with capital letter ending in “a”

```
def cap_end_a(word):
    return word[0].isupper() and word[-1] == "a"
```

- Example 2: words that contain a double letter

```
def contains_double_letter(word):
    last = word[0]
    for letter in word[1:]
        if last == letter:
            return True
        last = letter
    return False
```

Analyzing Words

- Example 1: words beginning with capital letter ending in "a"

```
def cap_end_a(word):
    return word[0].isupper() and word[-1] == "a"
```

- Example 2: words that contain a double letter

```
def contains_double_letter(word):
    for i in range(len(word)-1):
        if word[i] == word[i+1]:
            return True
    return False
```

Adding Statistics

- Example: let's count our special words

```
def count_words(f):
    count = count_cap_end_a = contains_double_letter = 0
    for line in f:
        for word in line.split():
            count = count + 1
            if cap_end_a(word):
                count_cap_end_a = count_cap_end_a + 1
            if contains_double_letter(word):
                count_double_letter = count_double_letter + 1
    print count, count_cap_end_a, count_double_letter
    print count_double_letter * 100 / count, "%"
```

Adding Statistics

- Example: let's count our special words

```
def count_words(f):
    count = count_cap_end_a = contains_double_letter = 0
    for line in f:
        for word in line.split():
            count += 1
            if cap_end_a(word):
                count_cap_end_a += 1
            if contains_double_letter(word):
                count_double_letter += 1
    print count, count_cap_end_a, count_double_letter
    print count_double_letter * 100 / count, "%"
```

Debugging by Testing Functions

- correct selection of tests important
- check obviously different cases for correct return value
- check corner cases (here: first letter, last letter etc.)
- Example:

```
def contains_double_letter(word):
    for i in range(len(word)-1):
        if word[i] == word[i+1]:
            return True
    return False
```

- test "mallorca" and "ibiza"
- test "llamada" and "bell"

LIST PROCESSING

Lists as Sequences

- lists are sequences of values
- lists can be constructed using "[" and "]"
- Example: [42, 23]
  ["Hello", "World", "!"]
  ["strings and", int, "mix", 2]
  []
- lists can be nested, i.e., a list can contain other lists
- Example: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
- lists are normal values, i.e., they can be printed, assigned etc.
- Example: x = [1, 2, 3]
  print x, [x, x], [[x, x], x]

Mutable Lists

- lists can be accessed using indices
- lists are mutable, i.e., they can be changed destructively
- Example:

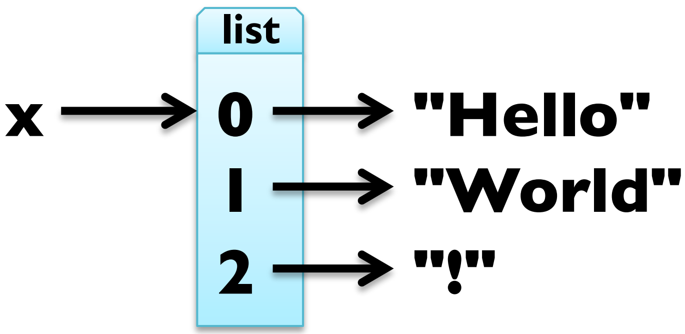
```
x = [1, 2, 3]
print x[1]
x[1] = 4
print x, x[1]
```

- len(object) and negative values work like for strings
- Example:

```
x[2] == x[-1]
x[1] == x[len(x)-2]
```

Stack Diagrams with Lists

- lists can be viewed as mappings from indices to elements
- Example 1: x = ["Hello", "World", "!"]



- Example 2: x = [[23, 42, -3.0], "Bye!"]

Traversing Lists

- for loop consecutively assigns variable to elements of list
- Example: print squares of numbers from 1 to 10

```
for x in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:
    print x**2
```

- arithmetic sequences can be generated using range function:
  - range([start,] stop[, step])
- Example:

```
range(4) == [0, 1, 2, 3]
range(1, 11) == [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
range(9, 1, -2) == [9, 7, 5, 3]
range(1,10, 2) == [1, 3, 5, 7, 9]
```

Traversing Lists

- for loop consecutively assigns variable to elements of list
- general form

```
for element in my_list:
    print element
```

- iteration through list with indices:

```
for index in range(len(my_list)):
    element = my_list[index]
    print element
```

- Example: in-situ update of list

```
x = [8388608, 43980465111104, 0.125]
for i in range(len(x)):
    x[i] = math.log(x[i], 2)
```

List Operations

- like for strings, "+" concatenates two lists
- Example:

```
[1, 2, 3] + [4, 5, 6] == range(1, 7)
[[23, 42] + [-3.0]] + ["Bye!"] == [[23, 42, -3.0], "Bye!"]
```

- like for strings, "* n" with integer n produces n copies
- Example:

```
len(["I", "love", "penguins!"] * 100) == 300
(range(1, 3) + range(3, 1, -1)) * 2 == [1, 2, 3, 2, 1, 2, 3, 2]
```

List Slices

- slices work just like for strings
- Example:

```
x = ["Hello", 2, "u", 2, "!"]
x[2:4] == ["u", 2]
x[2:] == x[-3:len(x)]
y = x[:]          # make a copy (lists are mutable!)
```

- BUT: we can also assign to slices!
- Example:

```
x[1:4] = ["to", "you", "too"]
x == ["Hello", "to", "you", "too", "!"]
x[1:3] = ["to me"]
x == ["Hello", "to me", "too", "!"]
x[2:3] = []
x == ["Hello", "to me", "!"]
```