DM536 / DM550 Part I
Introduction to Programming

Peter Schneider-Kamp

petersk@imada.sdu.dk

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

PROJECT PART I

Organizational Details

- 2 possible projects

- projects must be done individually, so no co-operation
- you may talk about the problem and ideas how to solve them

- deliverables:
  - Written 6 page report as specified in project description
  - handed in electronically as a PDF + source code files
  - pre-delivery deadline: October 2, 23:59
  - FINAL deadline: October 23, 23:59

- ENOUGH - now for the FUN part …

Fractals and the Beauty of Nature

- geometric objects similar to themselves at different scales

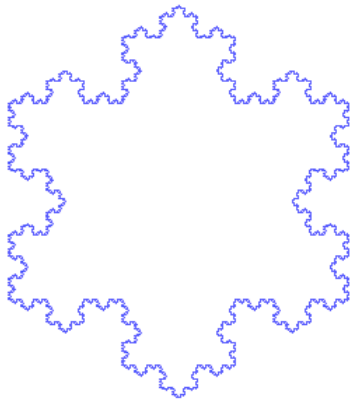
- many structures in nature are fractals:
  - snowflakes
  - lightning
  - ferns

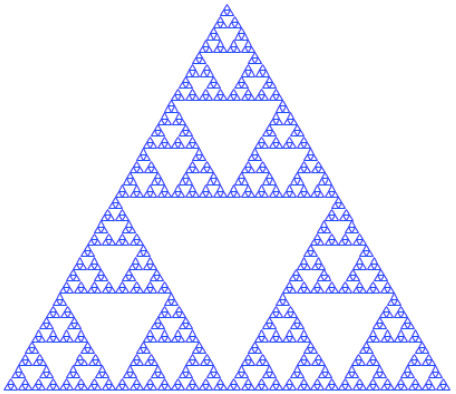
- **Goal:** generate fractals using Swampy

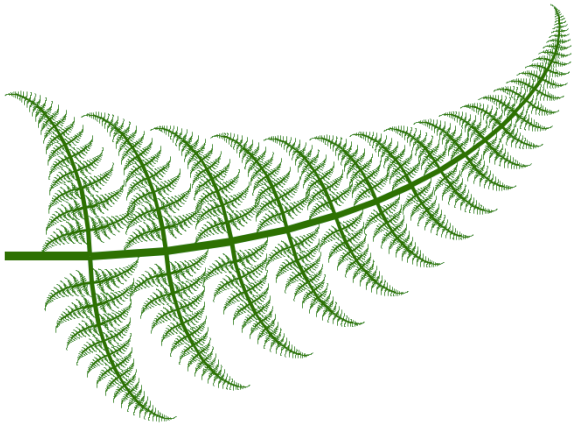
- **Challenges:** Recursion, Tuning, Library Use

Fractals and the Beauty of Nature

- Task 0: Preparation
  - understand implementation of Koch snowflake
- Task 1: Sierpinski Triangle
  - draw fractal triangle of fixed depth
- Task 2: Binary Tree
  - draw binary trees of fixed depth
- Task 3 (optional): Fern Time
  - draw beautiful fern leaves with fixed detail



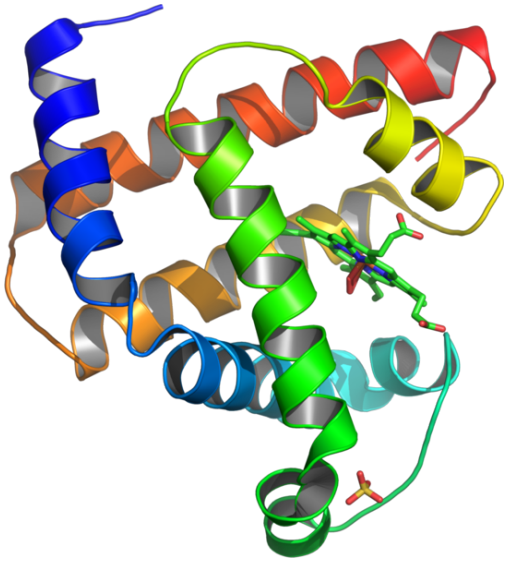




From DNA to Proteins

- proteins encoded by DNA base sequence using A, C, G, and T

- Background:
  - proteins are sequences of amino acids
  - amino acids encoded using three bases
  - chromosomes given as base sequences



- **Goal:** assemble and analyze sequences from files

- **Challenges:** File Handling, String and List Methods, Iteration

From DNA to Proteins

- Task 0: Preparation
  - download human DNA sequence and take a look at it
- Task 1: Assembling the Sequence
  - clean up the sequence and assemble it into one string
- Task 2: Finding Starting Points
  - find positions in string where ATG closely follows TATAAA
- Task 3: Finding End Points
  - find one of the potential end markers (TAG, TAA, TGA)
- Task 4 (optional): Potential Proteins without TATA Boxes
  - analysis of overlaps in encoded proteins

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

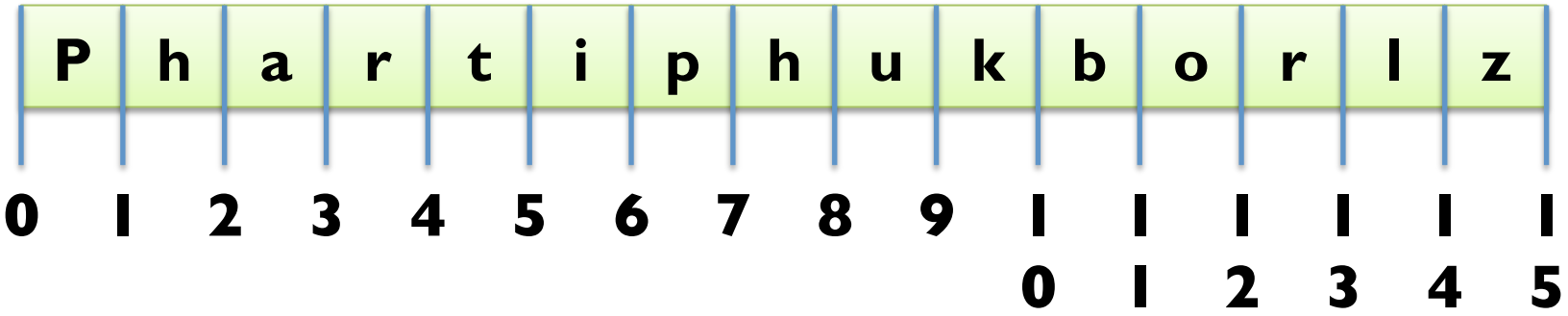
- slice = part of a string
- Example 1:

```
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:

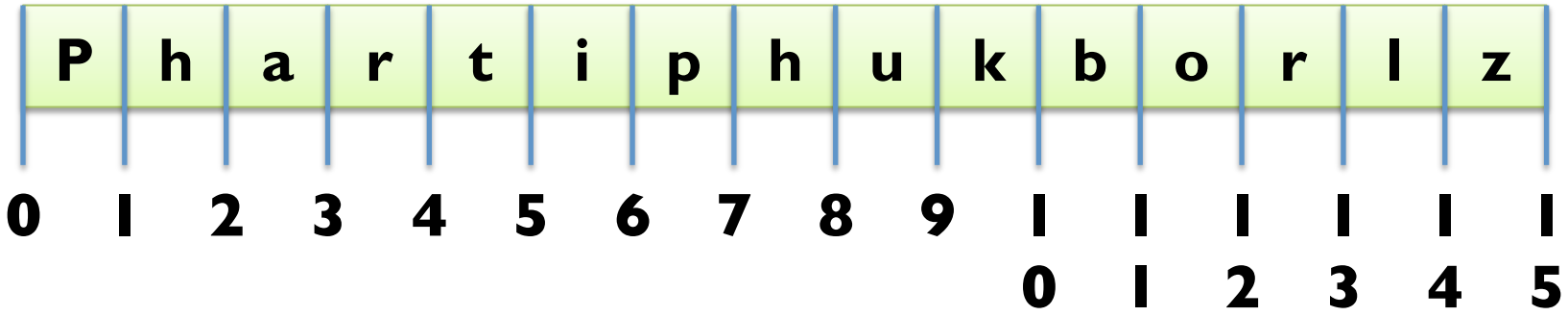


String Slices

- 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 = count_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 = count_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"