DM550/DM857
Introduction to Programming

Peter Schneider-Kamp

petersk@imada.sdu.dk

http://imada.sdu.dk/~petersk/DM550/

http://imada.sdu.dk/~petersk/DM857/

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:

  $\langle expr\rangle \;\Rightarrow\; \ldots \mid \langle expr_1\rangle[\langle expr_2\rangle]$

- $\langle expr_1\rangle$ = expression with value of type string
- index $\langle expr_2\rangle$ = 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

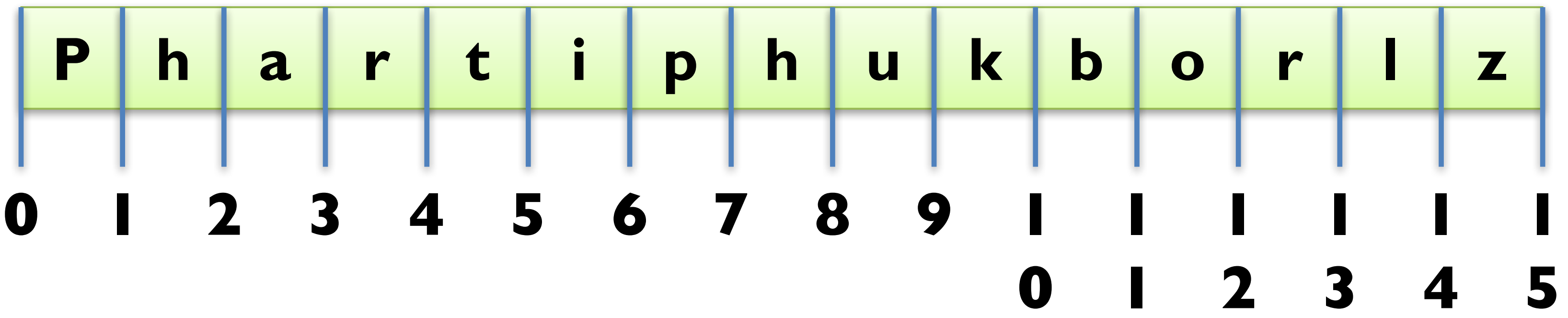
- slice = part of a string
- Example 1:

```
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:

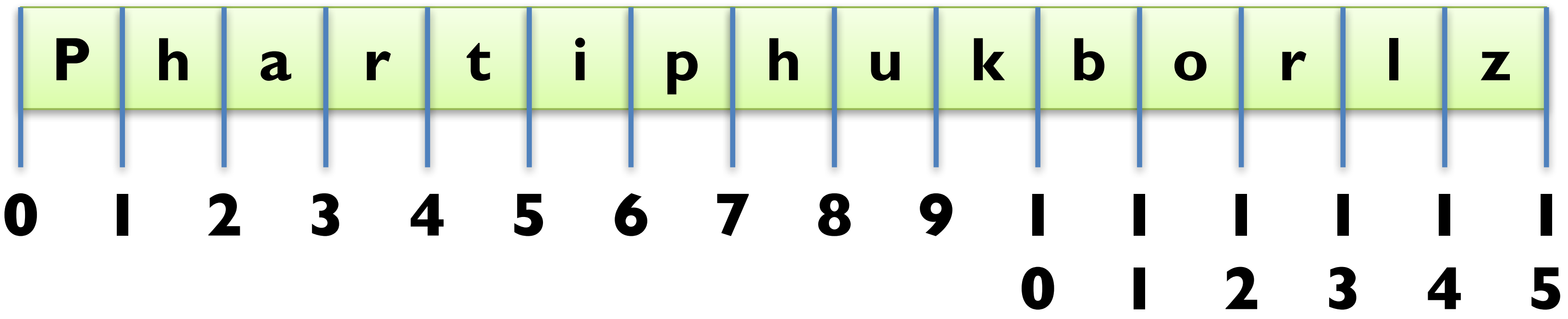


String Slices

- 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
```