Energy Informatics

System Design — Data Modeling

UNI FREIBURG

Albert-Ludwigs-Universität Freiburg

Peter Thiemann Gabriel Radanne

5 November 2018

Loops

Functions
Count occurrences of letter

UNI FREIBURG

Task

Write a function `count` that takes a string and a character and counts how often it occurs in the string.

Functions
Count occurrences of letter

Task

Write a function count that takes a string and a character and counts how often it occurs in the string.

Solution

```
>>> def count_element(str, ch):
...     count = 0
...     for c in str:
...         if c == ch:
...             count = count+1
...     return count
...
>>> count_element('atama', 'a')
3
>>> count_element('atama', 'x')
0
```

For loops

General form

```
for c in str:
    body
    :
```

- `c` must be a variable name
- `str` stands for a list or a string (for example)
- `body` and subsequent lines aligned with it are executed once for each element (character) of `str` from left to right
- variable `c` contains the current character

More generally

The same code works for other sequences

For example, for arrays

```
>>> count_element([1,2,3,2,1,2], 2)
3
>>> count_element([1,2,3,2,1,2], 4)
0
```

Summing the contents of an array

UNI FREIBURG

Task

Write a function `average` that takes an array with numbers and computes its arithmetic average.

Summing the contents of an array

Task

Write a function average that takes an array with numbers and computes its arithmetic average.

Solution

```
>>> def mysum(seq):   # predefined as sum
...     s = 0
...     for x in seq:
...         s += x
...     return s
...
>>> def average(seq):
...     return sum(seq) / len(seq)
```

Summing the contents of an array

Task

Write a function average that takes an array with numbers and computes its arithmetic average.

Solution

```
>>> def mysum(seq):   # predefined as sum
...     s = 0
...     for x in seq:
...         s += x
...     return s
...
>>> def average(seq):
...     return sum(seq) / len(seq)
```

Ok?

Missing a special case

What if `len(seq)==0`?

```
>>> average([])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in average
ZeroDivisionError: division by zero
```

Missing a special case

What if `len(seq)==0`?

```
>>> average([])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in average
ZeroDivisionError: division by zero
```

Safeguard such situations

Let's define that the average of an empty list is 0. This is an arbitrary choice, which is problem dependent.

```
def average0(seq):
    if len(seq) > 0:
        return sum(seq) / len(seq)
    else:
        return 0
```

Range

Task

Write a function that prints a table of the squares up to `n`.

Use `range`: an experiment

`range(b)` enumerates the elements of the list `[0, 1,..., b-1]`

```
>>> for i in range(8):
...     print ("{:5}{:5}".format(i, i*i))
...
    0    0
    1    1
    2    4
    3    9
    4   16
    5   25
    6   36
    7   49
```

How can we generalize?

How many positions are needed to print number n?

```
def positions(n):
    return math.floor(1 + math.log10(n))
```

How can we generalize?

How many positions are needed to print number *n*?

```
def positions(n):
    return math.floor(1 + math.log10(n))
```

How to create the format string?

```
p = 1 + positions(n*n)
f = "{{:{0}}}{{:{0}}}".format(p)
```

How can we generalize?

How many positions are needed to print number *n*?

```
def positions(n):
    return math.floor(1 + math.log10(n))
```

How to create the format string?

```
    p = 1 + positions(n*n)
    f = "{{:{0}}}{{:{0}}}".format(p)
```

Putting it all together

```
def squares(n):
    p = 1 + positions(n*n)
    f = "{{:{0}}}{{:{0}}}".format(p)
    for i in range(n):
        print(f.format(i, i*i))
```

More about ranges

UNI FREIBURG

`range(b)`

Enumerates `0`, `1`, `...`, `b-1`

More about ranges

UNI FREIBURG

`range(b)`

Enumerates `0`, `1`, `...`, `b-1`

`range(a,b)`

Enumerates `a`, `a+1`, `...`, `b-1`
Nothing if `a>=b`

More about ranges

UNI FREIBURG

`range(b)`

Enumerates `0`, `1`, `...`, `b-1`

`range(a,b)`

Enumerates `a`, `a+1`, `...`, `b-1`
Nothing if `a>=b`

`range(a,b,s)` for `s>0`

Enumerates `a`, `a+s`, `...`, `a+n*s`
where `n` is chosen maximal such that `a+n*s<b`
that is, if `s>0`, `n<(b-a)/s` which means
`n = math.floor ((b-a)/s)`

More about ranges

UNI FREIBURG

`range(b)`

Enumerates `0`, `1`, `...`, `b-1`

`range(a,b)`

Enumerates `a`, `a+1`, `...`, `b-1`
Nothing if `a>=b`

`range(a,b,s)` for `s>0`

Enumerates `a`, `a+s`, `...`, `a+n*s`
where `n` is chosen maximal such that `a+n*s<b`
that is, if `s>0`, `n<(b-a)/s` which means
`n = math.floor ((b-a)/s)`

Find out the story for `s<0` ...

Computing with lists

Dot product

```
def dotproduct(a, b):
    r = 0
    for i in range(min(len(a), len(b))):
        r += a[i]*b[i]
    return r
```

Compute the longest word in a text

UNI FREIBURG

Task

Given a text (as a string) find the longest word in it.

Compute the longest word in a text

UNI FREIBURG

Task

Given a text (as a string) find the longest word in it.

Subtasks

1. find all words in a string (result: a list)
2. find the longest word in a list

Dictionaries

UNI FREIBURG

Special datatype in scripting languages

- A dictionary stores an association between **keys** and **values**.
- Strings and numbers can serve as keys (among others).

Dictionaries

Special datatype in scripting languages

- A dictionary stores an association between **keys** and **values**.
- Strings and numbers can serve as keys (among others).

Talking to Python

```
>>> tel = { "gl": 8121, "cs": 8181 }
>>> tel["pt"] = 8051
>>> tel['cs']
8181
>>> del tel['cs']
>>> tel
{'gl': 8121, 'pt': 8051}
>>> tel['cs']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'cs'
```

Application of dictionaries

UNI FREIBURG

Task

Count the number of occurrences of all letters in a string.

Application of dictionaries

Task

Count the number of occurrences of all letters in a string.

Python source

```
def count_all_letters(s):
    d = dict(); # empty dictionary
    for c in s:
        d[c] = d[c] + 1 if c in d else 1
    return d
```

- Before using `d[c]`, we need to check whether `c in d`, that is, whether `c` is a defined key in dictionary `d`

Example and Alternative implementation

Example uses

```
>>> count_all_letters("atama")
{'a': 3, 'm': 1, 't': 1}
>>> count_all_letters(
...     "the␣world␣is␣what␣you␣think␣it␣is")
{'t': 4, 'h': 3, 'e': 1, '␣': 7, 'w': 2, 'o': 2, 'r': 1,
```

Example and Alternative implementation

Example uses

```
>>> count_all_letters("atama")
{'a': 3, 'm': 1, 't': 1}
>>> count_all_letters(
...     "the world is what you think it is")
{'t': 4, 'h': 3, 'e': 1, ' ': 7, 'w': 2, 'o': 2, 'r': 1,
```

Alternative implementation

```
def count_letters(s):
    d = {}
    for c in s:
        if c in d:
            d[c] = d[c] + 1
        else:
            d[c] = 1
    return d
```

Generalizing

UNI FREIBURG

N.B.

The code for `count_all_letters` does not depend on strings or letters. It can be used generally to collect the count of all different elements of a sequence. Examples for sequences:

- Strings — letter count
- List of numbers
- List of words — word count

End Part III

Iterating on dictionaries

UNI FREIBURG

Task

Create a dictionary that groups letters by their number of occurrences.

Iterating on dictionaries

Task

Create a dictionary that groups letters by their number of occurrences.

Example uses

```
>>> d = count_all_letters(
        "the␣world␣is␣what␣you␣think␣it␣is")
>>> by_count(d)
{4: ['t', 'i'],
 3: ['h'],
 1: ['e', 'r', 'l', 'd', 'a', 'y', 'u', 'n', 'k'],
 7: ['␣'],
 2: ['w', 'o', 's']}
```

Iterating on dictionaries

UNI FREIBURG

Implementation

```
def by_count(d):
    count = dict()
    for k, n in d.items():
      if n in count:
        count[n] = count[n] + [k]
      else:
        count[n] = [k]
    return count
```