Outline

1. Modular Design
   multiple choice quiz
   bottom-up design

2. Python Implementation
   the modules
   main program: quiz.py
   namespaces in Python

3. The Software Cycle
   quality of product and process
   waterfall and spiral model

4. Summary + Assignments
modular design
software cycle & quality

1. Modular Design
   multiple choice quiz
   bottom-up design

2. Python Implementation
   the modules
   main program: quiz.py
   namespaces in Python

3. The Software Cycle
   quality of product and process
   waterfall and spiral model

4. Summary + Assignments
Multiple Choice Quiz

problem specification

To test our knowledge of computer science concepts, we would like an automatic multiple choice quiz.

Our friend the computer will

1. show a question with multiple possible answers
2. wait for us to make a choice
3. immediately evaluate our choice
4. ask us if we want to continue
5. at the end print how well we did

Wanted: a small but extendable program.
To test our knowledge of computer science concepts, we would like an automatic multiple choice quiz.

Our friend the computer will

1. show a question with multiple possible answers
2. wait for us to make a choice
3. immediately evaluate our choice
4. ask us if we want to continue
5. at the end print how well we did

Wanted: a small but extendable program.
To test our knowledge of computer science concepts, we would like an automatic multiple choice quiz.

Our friend the computer will

1. show a question with multiple possible answers
2. wait for us to make a choice
3. immediately evaluate our choice
4. ask us if we want to continue
5. at the end print how well we did

Wanted: a small but extendable program.
Multiple Choice Quiz

problem specification

To test our knowledge of computer science concepts, we would like an automatic multiple choice quiz.

Our friend the computer will

1. show a question with multiple possible answers
2. wait for us to make a choice
3. immediately evaluate our choice
4. ask us if we want to continue
5. at the end print how well we did

Wanted: a small but extendable program.
Multiple Choice Quiz

problem specification

To test our knowledge of computer science concepts, we would like an automatic multiple choice quiz.

Our friend the computer will

1. show a question with multiple possible answers
2. wait for us to make a choice
3. immediately evaluate our choice
4. ask us if we want to continue
5. at the end print how well we did

Wanted: a small but extendable program.
Multiple Choice Quiz
problem specification

To test our knowledge of computer science concepts, we would like an automatic multiple choice quiz.

Our friend the computer will

1. show a question with multiple possible answers
2. wait for us to make a choice
3. immediately evaluate our choice
4. ask us if we want to continue
5. at the end print how well we did

Wanted: a small but extendable program.
To test our knowledge of computer science concepts, we would like an automatic multiple choice quiz.

Our friend the computer will

1. show a question with multiple possible answers
2. wait for us to make a choice
3. immediately evaluate our choice
4. ask us if we want to continue
5. at the end print how well we did

Wanted: a small but extendable program.
The main program **quiz.py**, executed at the prompt $:$

$ python quiz.py

The octal system has base ...

1. 16
2. 8
3. 10
4. 2

Type a number between 1 and 4 : 2

This is the correct answer

Continue ? (y/n) n

#correct : 1 #wrong : 0 → 100.00%
Tools for Use
what does Python provide?

Python provides

1️⃣ a command line or Python shell environment
2️⃣ data structures: lists, tuples, and dictionaries
3️⃣ if-elif-else, loops, and functions
4️⃣ random number generators in the module `random`
5️⃣ our modules will reflect our bottom-up design

What is there at the bottom of our program?
Tools for Use
what does Python provide?

Python provides

① a command line or Python shell environment
② data structures: lists, tuples, and dictionaries
③ if-elif-else, loops, and functions
④ random number generators in the module `random`
⑤ our modules will reflect our bottom-up design

What is there at the bottom of our program?
Python provides

1. a command line or Python shell environment
2. data structures: lists, tuples, and dictionaries
3. if-elif-else, loops, and functions
4. random number generators in the module `random`
5. our modules will reflect our bottom-up design

What is there at the bottom of our program?
Tools for Use
what does Python provide?

Python provides

1. a command line or Python shell environment
2. data structures: lists, tuples, and dictionaries
3. if-elif-else, loops, and functions
4. random number generators in the module `random`
5. our modules will reflect our bottom-up design

What is there at the bottom of our program?
Tools for Use
what does Python provide?

Python provides

① a command line or Python shell environment
② data structures: lists, tuples, and dictionaries
③ if-elif-else, loops, and functions
④ random number generators in the module `random`
⑤ our modules will reflect our bottom-up design

What is there at the bottom of our program?
Tools for Use
what does Python provide?

Python provides

1. a command line or Python shell environment
2. data structures: lists, tuples, and dictionaries
3. if-elif-else, loops, and functions
4. random number generators in the module `random`
5. our modules will reflect our bottom-up design

What is there at the bottom of our program?
Modular Design
multiple choice quiz
bottom-up design

Python Implementation
the modules
main program: quiz.py
namespaces in Python

The Software Cycle
quality of product and process
waterfall and spiral model

Summary + Assignments
The Bottom of our Program

A multiple choice quiz consists of

1. questions that have short answers;
2. answers, correct ones and variations.

Every question and answer will be a string.

Dictionaries seem very appropriate:

1. the questions are numbered, key = number, the values are strings
2. answers for the corresponding question have the same key as the question
3. answers are dictionaries of dictionaries
4. convention: correct answer comes first
The Bottom of our Program

A multiple choice quiz consists of

1. questions that have short answers;
2. answers, correct ones and variations.

Every question and answer will be a string.

Dictionaries seem very appropriate:

1. the questions are numbered, key = number, the values are strings
2. answers for the corresponding question have the same key as the question
3. answers are dictionaries of dictionaries
4. convention: correct answer comes first
The Bottom of our Program

A multiple choice quiz consists of

1. questions that have short answers;
2. answers, correct ones and variations.

Every question and answer will be a string.

Dictionaries seem very appropriate:

1. the questions are numbered, key = number, the values are strings
2. answers for the corresponding question have the same key as the question
3. answers are dictionaries of dictionaries
4. convention: correct answer comes first
The Bottom of our Program

A multiple choice quiz consists of

1. questions that have short answers;
2. answers, correct ones and variations.

Every question and answer will be a string.

Dictionaries seem very appropriate:

1. the questions are numbered, key = number, the values are strings
2. answers for the corresponding question have the same key as the question
3. answers are dictionaries of dictionaries
4. convention: correct answer comes first
The Bottom of our Program

A multiple choice quiz consists of

1. questions that have short answers;
2. answers, correct ones and variations.

Every question and answer will be a string.

Dictionaries seem very appropriate:

1. the questions are numbered, key = number, the values are strings
2. answers for the corresponding question have the same key as the question
3. answers are dictionaries of dictionaries
4. convention: correct answer comes first
The Bottom of our Program

A multiple choice quiz consists of

1. questions that have short answers;
2. answers, correct ones and variations.

Every question and answer will be a string.

Dictionaries seem very appropriate:

1. the questions are numbered, key = number, the values are strings
2. answers for the corresponding question have the same key as the question
3. answers are dictionaries of dictionaries
4. convention: correct answer comes first
The Other Modules

We separate the functionalities:

1. generating questions and evaluating answers;
2. showing questions and asking answers.

Two modules of functions: questions and dialogue.

Both modules import the Q&A dictionary and export their functions to the main program.

The formulation of questions and answers is centralized in the Q&A dictionary.

A multiple choice question then consists of

1. the number of the question;
2. a permutation of the answers.

In Python we use a tuple, e.g.: (3,[2,1,3,4]).
The Other Modules

We separate the functionalities:

1. generating questions and evaluating answers;
2. showing questions and asking answers.

Two modules of functions: questions and dialogue.

Both modules import the Q&A dictionary and export their functions to the main program.

The formulation of questions and answers is centralized in the Q&A dictionary.

A multiple choice question then consists of

1. the number of the question;
2. a permutation of the answers.

In Python we use a tuple, e.g.: (3,[2,1,3,4]).
The Other Modules

We separate the functionalities:

1. generating questions and evaluating answers;
2. showing questions and asking answers.

Two modules of functions: questions and dialogue.

Both modules import the Q&A dictionary and export their functions to the main program.

The formulation of questions and answers is centralized in the Q&A dictionary.

A multiple choice question then consists of

1. the number of the question;
2. a permutation of the answers.

In Python we use a tuple, e.g.: (3,[2,1,3,4]).
The Other Modules

We separate the functionalities:

1. generating questions and evaluating answers;
2. showing questions and asking answers.

Two modules of functions: questions and dialogue.

Both modules import the Q&A dictionary and export their functions to the main program.

The formulation of questions and answers is centralized in the Q&A dictionary.

A multiple choice question then consists of

1. the number of the question;
2. a permutation of the answers.

In Python we use a tuple, e.g.: (3,[2,1,3,4]).
The Other Modules

We separate the functionalities:

1. generating questions and evaluating answers;
2. showing questions and asking answers.

Two modules of functions: questions and dialogue.

Both modules import the Q&A dictionary and export their functions to the main program.

The formulation of questions and answers is centralized in the Q&A dictionary.

A multiple choice question then consists of

1. the number of the question;
2. a permutation of the answers.

In Python we use a tuple, e.g.: (3,[2,1,3,4]).
The Other Modules

We separate the functionalities:

1. generating questions and evaluating answers;
2. showing questions and asking answers.

Two modules of functions: questions and dialogue.

Both modules import the Q&A dictionary and export their functions to the main program.

The formulation of questions and answers is centralized in the Q&A dictionary.

A multiple choice question then consists of

1. the number of the question;
2. a permutation of the answers.

In Python we use a tuple, e.g.: (3,[2,1,3,4]).
At the bottom: the Q&A dictionary.
Two libraries of functions: questions and dialogue.

Information Hiding:
the main program does not import the Q&A dictionary.
The Modular Design

At the bottom: the Q&A dictionary.
Two libraries of functions: questions and dialogue.

Information Hiding:
the main program does not import the Q&A dictionary.
The Modular Design

At the bottom: the Q&A dictionary.
Two libraries of functions: questions and dialogue.

Information Hiding:
the main program does not import the Q&A dictionary.
The Modular Design

At the bottom: the Q&A dictionary.
Two libraries of functions: questions and dialogue.

Information Hiding:
the main program does not import the Q&A dictionary.
The Modular Design

At the bottom: the Q&A dictionary.
Two libraries of functions: questions and dialogue.

Information Hiding:
the main program does not import the Q&A dictionary.
modular design
software cycle & quality

1. Modular Design
   multiple choice quiz
   bottom-up design

2. Python Implementation
   the modules
   main program: quiz.py
   namespaces in Python

3. The Software Cycle
   quality of product and process
   waterfall and spiral model

4. Summary + Assignments
The Q&A Dictionary
in the file quand.py

questions = {
    1: 'How many bits go in one byte ?',
    2: 'Who is the main developer of Python ?',
    3: 'Which statement is true ?',
    4: 'The octal system has base ...'
}

answers = {
    2: { 1: 'Guido van Rossum', 2: 'John von Neumann',
         3: 'Rashi Gupta', 4: 'John Backus' },
    3: { 1: 'a compiler translates source code',
         2: 'a compiler executes object code',
         3: 'a compiler formats source code',
         4: 'a compiler stores code on file' },
    4: { 1: '8', 2: '2', 3: '10', 4: '16' }
}
The Q&A Dictionary
in the file quand.py

questions = { \
    1:'How many bits go in one byte ?', \
    2:'Who is the main developer of Python ?', \
    3:'Which statement is true ?', \
    4:'The octal system has base ...' \
}

answers = { \
    1:{ 1:'8', 2:'4', 3:'2', 4:'16' }, \
    2:{ 1:'Guido van Rossum', 2:'John von Neumann', 
            3:'Rashi Gupta', 4:'John Backus' }, \
    3:{ 1:'a compiler translates source code', 
            2:'a compiler executes object code', 
            3:'a compiler formats source code', 
            4:'a compiler stores code on file' }, \
    4:{ 1:'8', 2:'2', 3:'10', 4:'16' } \
}
Specification of questions.py

help(questions) shows under FUNCTIONS:

evaluate(L, c)
    evaluates choice c to True or False
    where L is a list of answers

generate()
    returns a multiple choice question
    as a tuple (q,L), where q is a number,
    and L is a list of numbers with answers

shuffle(L)
    shuffles the order of the answers
    on return is a permutation of the list L

update_tally(t, b)
    updates tally of True or False answers
    where t is a list of two elements and
    b is True or False, returns updated tally
Specification of questions.py

help(questions) shows under FUNCTIONS:

evaluate(L, c)
  evaluates choice c to True or False
  where L is a list of answers

generate()
  returns a multiple choice question
  as a tuple (q,L), where q is a number,
  and L is a list of numbers with answers

shuffle(L)
  shuffles the order of the answers
  on return is a permutation of the list L

update_tally(t, b)
  updates tally of True or False answers
  where t is a list of two elements and
  b is True or False, returns updated tally
Specification of questions.py

help(questions) shows under FUNCTIONS:

evaluate(L, c)
  evaluates choice c to True or False
  where L is a list of answers

generate()
  returns a multiple choice question
  as a tuple (q,L), where q is a number,
  and L is a list of numbers with answers

shuffle(L)
  shuffles the order of the answers
  on return is a permutation of the list L

update_tally(t, b)
  updates tally of True or False answers
  where t is a list of two elements and
  b is True or False, returns updated tally
Specification of questions.py

`help(questions)` shows under FUNCTIONS:

```python
evaluate(L, c)
    evaluates choice c to True or False
    where L is a list of answers

generate()
    returns a multiple choice question
    as a tuple (q,L), where q is a number,
    and L is a list of numbers with answers

shuffle(L)
    shuffles the order of the answers
    on return is a permutation of the list L

update_tally(t, b)
    updates tally of True or False answers
    where t is a list of two elements and
    b is True or False, returns updated tally
```
Specification of dialogue.py

`help(dialogue)` shows under FUNCTIONS:

- `prompt_choice(n)`
  - asks the user for a choice
  - where n is the number of choices

- `show_outcome(b)`
  - reports outcome to user
  - where b is True or False

- `show_question(q, L)`
  - displays the questions and the answers
  - on input is the number q of the question
  - and the list L of answers

- `show_tally(t)`
  - shows final tally, t is list
**Specification of dialogue.py**

`help(dialogue)` shows under FUNCTIONS:

- `prompt_choice(n)` asks the user for a choice where `n` is the number of choices
- `show_outcome(b)` reports outcome to user where `b` is True or False
- `show_question(q, L)` displays the questions and the answers on input is the number `q` of the question and the list `L` of answers
- `show_tally(t)` shows final tally, `t` is list
Specification of dialogue.py

```python
help(dialogue) shows under FUNCTIONS:

  prompt_choice(n)
  asks the user for a choice
  where n is the number of choices

  show_outcome(b)
  reports outcome to user
  where b is True or False

  show_question(q, L)
  displays the questions and the answers
  on input is the number q of the question
  and the list L of answers

  show_tally(t)
  shows final tally, t is list
```
Specification of dialogue.py

**help(dialogue)** shows under **FUNCTIONS:**

- **prompt_choice(n)**
  - asks the user for a choice
  - where n is the number of choices

- **show_outcome(b)**
  - reports outcome to user
  - where b is True or False

- **show_question(q, L)**
  - displays the questions and the answers
  - on input is the number q of the question
  - and the list L of answers

- **show_tally(t)**
  - shows final tally, t is list
modular design
software cycle & quality

1. Modular Design
   multiple choice quiz
   bottom-up design

Python Implementation
   the modules
   main program: quiz.py
   namespaces in Python

2. **Python Implementation**
   the modules
   main program: quiz.py
   namespaces in Python

3. The Software Cycle
   quality of product and process
   waterfall and spiral model

4. Summary + Assignments
import questions
import dialogue

t = [0,0]
while True:
    (q,a) = questions.generate()
    s = questions.shuffle(a)
    dialogue.show_question(q,s)
    c = dialogue.prompt_choice(len(s))
    e = questions.evaluate(s,c)
    dialogue.show_outcome(e)
    t = questions.update_tally(t,e)
    more = raw_input(‘continue ? (y/n) ’)
    if more != ’y’: break

dialogue.show_tally(t)
import questions
import dialogue

t = [0,0]
while True:
    (q,a) = questions.generate()
    s = questions.shuffle(a)
    dialogue.show_question(q,s)
    c = dialogue.prompt_choice(len(s))
    e = questions.evaluate(s,c)
    dialogue.show_outcome(e)
    t = questions.update_tally(t,e)
    more = raw_input('continue ? (y/n) ')
    if more != 'y': break
    dialogue.show_tally(t)
import questions
import dialogue

t = [0, 0]
while True:
    (q,a) = questions.generate()
    s = questions.shuffle(a)
    dialogue.show_question(q,s)
    c = dialogue.prompt_choice(len(s))
    e = questions.evaluate(s,c)
    dialogue.show_outcome(e)
    t = questions.update_tally(t,e)
    more = raw_input('continue ? (y/n) ')
    if more != 'y': break

dialogue.show_tally(t)
import questions
import dialogue

t = [0, 0]
while True:
    (q,a) = questions.generate()
    s = questions.shuffle(a)
    dialogue.show_question(q,s)
    c = dialogue.prompt_choice(len(s))
    e = questions.evaluate(s,c)
    dialogue.show_outcome(e)
    t = questions.update_tally(t,e)
    more = raw_input('continue ? (y/n) ')
    if more != 'y': break

dialogue.show_tally(t)
import questions
import dialogue

t = [0, 0]
while True:
    (q,a) = questions.generate()
    s = questions.shuffle(a)
    dialogue.show_question(q,s)
    c = dialogue.prompt_choice(len(s))
    e = questions.evaluate(s,c)
    dialogue.show_outcome(e)
    t = questions.update_tally(t,e)
    more = raw_input('continue ? (y/n) ')
    if more != 'y': break

dialogue.show_tally(t)
Two Functions in questions.py

```python
import quand

def generate():
    
    """
    returns a multiple choice question
    as a tuple (q,L), where q is a number,
    and L is a list of numbers with answers
    """
    import random
    q = random.randint(1,len(quand.questions))
    L = range(1,len(quand.answers[q])+1)
    return (q,L)

def evaluate(L, c):
    """
    evaluates choice c to True or False
    where L is a list of answers
    """
    return (L[c-1] == 1)
```
import quand

def generate():
    """
    returns a multiple choice question as a tuple (q,L), where q is a number, and L is a list of numbers with answers
    """
    import random
    q = random.randint(1,len(quand.questions))
    L = range(1,len(quand.answers[q])+1)
    return (q,L)

def evaluate(L,c):
    """
    evaluates choice c to True or False where L is a list of answers
    """
    return (L[c-1] == 1)
show_question() in dialogue.py

```python
import quand

def show_question(q,L):
    """
    displays the questions and the answers
    on input is the number q of the question
    and the list L of answers
    """
    print quand.questions[q]
    choice = 1
    for answer in L:
        p = '%d' % choice + ' . '
        p = p + quand.answers[q][answer]
        print p
    choice = choice + 1
```
Giving Feedback

def show_outcome(b):
    """
    reports outcome to user
    where b is True or False
    """
    if b:
        print 'this is the correct answer'
    else:
        print 'wrong, please try again'

def show_tally(t):
    """
    shows final tally, t is list
    """
    s = '#correct : %d' % t[1]
    s += ' #wrong : %d' % t[0]
    f = 100.0*t[1]/sum(t)
    s += ' -> %.2f%%' % f
    print s
Giving Feedback

def show_outcome(b):
    
    reports outcome to user
    where b is True or False
    
    if b:
        print 'this is the correct answer'
    else:
        print 'wrong, please try again'

def show_tally(t):
    
    shows final tally, t is list
    
    s = '#correct : %d' % t[1]
s += ' #wrong : %d' % t[0]
f = 100.0*t[1]/sum(t)
s += ' -> %.2f%%' % f
print s
modular design
software cycle & quality

1. Modular Design
   multiple choice quiz
   bottom-up design

2. Python Implementation
   the modules
   main program: quiz.py
   namespaces in Python

3. The Software Cycle
   quality of product and process
   waterfall and spiral model

4. Summary + Assignments
A **namespace** is a collection of names available to use. Importing from modules enlarges the current namespace. `dir()` returns a list of the attributes of the module

```python
>>> import random
>>> dir(random)
[ ... , 'uniform', ... ]
>>> import numpy.random
>>> dir(numpy.random)
[ ... , 'uniform', ... ]
```

There are two **uniform random number generators**: `random.uniform()` and `numpy.random.uniform()`.

```python
>>> from random import uniform as u1
>>> from numpy.random import uniform as u2
u1() and u2 now belong to the current namespace.
```
Namespaces in Python

A **namespace** is a collection of names available to use. Importing from modules enlarges the current namespace. `dir()` returns a list of the attributes of the module.

```python
>>> import random
>>> dir(random)
[ ... , 'uniform', ... ]
>>> import numpy.random
>>> dir(numpy.random)
[ ... , 'uniform', ... ]
```

There are two **uniform random number generators**: `random.uniform()` and `numpy.random.uniform()`.

```python
>>> from random import uniform as u1
>>> from numpy.random import uniform as u2
u1() and u2 now belong to the current namespace
```
Namespaces in Python

A **namespace** is a collection of names available to use. Importing from modules enlarges the current namespace. `dir()` returns a list of the attributes of the module.

```python
>>> import random
>>> dir(random)
[ ... , 'uniform', ... ]
>>> import numpy.random
>>> dir(numpy.random)
[ ... , 'uniform', ... ]
```

There are two uniform random number generators: `random.uniform()` and `numpy.random.uniform()`.

```python
>>> from random import uniform as u1
>>> from numpy.random import uniform as u2
```

`u1()` and `u2` now belong to the current namespace.
Namespaces in Python

A namespace is a collection of names available to use. Importing from modules enlarges the current namespace. `dir()` returns a list of the attributes of the module.

```python
>>> import random
>>> dir(random)
[ ... , 'uniform', ... ]
>>> import numpy.random
>>> dir(numpy.random)
[ ... , 'uniform', ... ]
```

There are two uniform random number generators: `random.uniform()` and `numpy.random.uniform()`.

```python
>>> from random import uniform as u1
>>> from numpy.random import uniform as u2
```

`u1()` and `u2` now belong to the current namespace.
The Software Cycle

We are experiencing the software crisis:

→ the industrial sector can not keep up with the ever growing demands of the market.

Symptoms:

1. products are delivered late
2. the cost exceed budgets
3. too many bugs appear

The relative cost of hardware versus software

- was 80% versus 20% in the 1960s
- became 20% versus 80% in the 1980s
- keeps going in the same direction.

Reason: hardware production is technology intensive, whereas software creation is human intensive.
The Software Cycle

We are experiencing the software crisis: the industrial sector can not keep up with the ever growing demands of the market.

Symptoms:

1. products are delivered late
2. the cost exceed budgets
3. too many bugs appear

The relative cost of hardware versus software

- was 80% versus 20% in the 1960s
- became 20% versus 80% in the 1980s
- keeps going in the same direction.

Reason: hardware production is technology intensive, whereas software creation is human intensive.
The Software Cycle

We are experiencing the software crisis:
→ the industrial sector can not keep up with the ever growing demands of the market.

Symptoms:

1. products are delivered late
2. the cost exceed budgets
3. too many bugs appear

The relative cost of hardware versus software

- was 80% versus 20% in the 1960s
- became 20% versus 80% in the 1980s
- keeps going in the same direction.

Reason: hardware production is technology intensive, whereas software creation is human intensive.
The Software Cycle

We are experiencing the software crisis:
→ the industrial sector can not keep up with the ever growing demands of the market.

Symptoms:

1. products are delivered late
2. the cost exceed budgets
3. too many bugs appear

The relative cost of hardware versus software

- was 80% versus 20% in the 1960s
- became 20% versus 80% in the 1980s
- keeps going in the same direction.

Reason: hardware production is technology intensive, whereas software creation is human intensive.
The Software Cycle

We are experiencing the software crisis:
→ the industrial sector can not keep up with the ever growing demands of the market.

Symptoms:

1. products are delivered late
2. the cost exceed budgets
3. too many bugs appear

The relative cost of hardware versus software

- was 80% versus 20% in the 1960s
- became 20% versus 80% in the 1980s
- keeps going in the same direction.

Reason: hardware production is technology intensive, whereas software creation is human intensive.
The Software Cycle

We are experiencing the software crisis:
→ the industrial sector can not keep up with the ever growing demands of the market.

Symptoms:

1. products are delivered late
2. the cost exceed budgets
3. too many bugs appear

The relative cost of hardware versus software

• was 80% versus 20% in the 1960s
• became 20% versus 80% in the 1980s
• keeps going in the same direction.

Reason: hardware production is technology intensive, whereas software creation is human intensive.
modular design
software cycle & quality

1. Modular Design
   multiple choice quiz
   bottom-up design

2. Python Implementation
   the modules
   main program: quiz.py
   namespaces in Python

3. The Software Cycle
   quality of product and process
   waterfall and spiral model

4. Summary + Assignments
Quality of Software Design

Industrial quality refers to products

1. that must meet a target value on average,
2. and within a tolerable standard deviation.

Example: a search engine should give one page of relevant answers within an average of 10 seconds, and standard deviation of 2 seconds.

We distinguish between quality of product and process:

1. a high quality product meets user’s expectations,
2. also the quality of the production process matters!

External qualities can be observed directly by the user. Example: easy to use

Internal qualities can be measured only by the producer. Example: design documentation
Quality of Software Design

industrial quality

Industrial quality refers to products

1. that must meet a target value on average,
2. and within a tolerable standard deviation.

example: a search engine should give one page of relevant answers within an average of 10 seconds, and standard deviation of 2 seconds.

We distinguish between quality of product and process:

1. a high quality product meets user’s expectations,
2. also the quality of the production process matters!

External qualities can be observed directly by the user.

example: easy to use

Internal qualities can be measured only by the producer.

example: design documentation
Quality of Software Design

industrial quality

Industrial quality refers to products

1. that must meet a target value on average,
2. and within a tolerable standard deviation.

example: a search engine should give one page of relevant answers within an average of 10 seconds, and standard deviation of 2 seconds.

We distinguish between quality of product and process:

1. a high quality product meets user’s expectations,
2. also the quality of the production process matters!

External qualities can be observed directly by the user.
example: easy to use

Internal qualities can be measured only by the producer.
example: design documentation
Quality of Software Design

industrial quality

Industrial quality refers to products

1. that must meet a target value on average,
2. and within a tolerable standard deviation.

example: a search engine should give one page of relevant answers within an average of 10 seconds, and standard deviation of 2 seconds.

We distinguish between quality of product and process:

1. a high quality product meets user’s expectations,
2. also the quality of the production process matters!

External qualities can be observed directly by the user.
example: easy to use

Internal qualities can be measured only by the producer.
example: design documentation
Quality of Software Design

industrial quality

Industrial quality refers to products

1. that must meet a target value on average,
2. and within a tolerable standard deviation.

*example:* a search engine should give one page of relevant answers within an average of 10 seconds, and standard deviation of 2 seconds.

We distinguish between quality of product and process:

1. a high quality product meets user’s expectations,
2. also the quality of the production process matters!

External qualities can be observed directly by the user. *example:* easy to use

Internal qualities can be measured only by the producer. *example:* design documentation
Quality of Software Design

Industrial quality refers to products

1. that must meet a target value on average,
2. and within a tolerable standard deviation.

*example*: a search engine should give one page of relevant answers within an average of 10 seconds, and standard deviation of 2 seconds.

We distinguish between quality of product and process:

1. a high quality product meets user’s expectations,
2. also the quality of the production process matters!

External qualities can be observed directly by the user.

*example*: easy to use

Internal qualities can be measured only by the producer.

*example*: design documentation
Software Quality Features
what is good software?

A short list of important criteria:

**correctness** must satisfy requirements

*How reliable is the software?*

**efficiency** economical use of resources such as hardware and time

*Do we understand how the complexity of the problem and the software relate?*

**modifiable** to different environments and requirements

*How hard is it to correct errors or to add new features?*

**reusability** of the same software in different applications

*Mathematical libraries are a typical example of reusable software.*
Software Quality Features

what is good software?

A short list of important criteria:

**correctness** must satisfy requirements

How reliable is the software?

**efficiency** economical use of resources such as hardware and time

Do we understand how the complexity of the problem and the software relate?

**modifiability** to different environments and requirements

How hard is it to correct errors or to add new features?

**reusability** of the same software in different applications

Mathematical libraries are a typical example of reusable software.
Software Quality Features

what is good software?

A short list of important criteria:

**correctness** must satisfy requirements

*How reliable is the software?*

**efficiency** economical use of resources such as hardware and time

*Do we understand how the complexity of the problem and the software relate?*

**modifiability** to different environments and requirements

*How hard is it to correct errors or to add new features?*

**reusability** of the same software in different applications

*Mathematical libraries are a typical example of reusable software.*
Software Quality Features
what is good software?

A short list of important criteria:

**correctness**
must satisfy requirements

*How reliable is the software?*

**efficiency**
economical use of resources such as hardware and time

*Do we understand how the complexity of the problem and the software relate?*

**modifiability**
to different environments and requirements

*How hard is it to correct errors or to add new features?*

**reusability**
of the same software in different applications

*Mathematical libraries are a typical example of reusable software.*
1. Modular Design
   multiple choice quiz
   bottom-up design

2. Python Implementation
   the modules
   main program: quiz.py
   namespaces in Python

3. The Software Cycle
   quality of product and process
   waterfall and spiral model

4. Summary + Assignments
The Waterfall Model
seven phases in sequence

The software life cycle model describes the software production in seven phases:

1. feasibility study: cost-benefit analysis
2. system requirements specification: what must it do?
3. system architecture design: modules and relations
4. implementation and verification of modules
5. system integration and verification
6. installation: deliver software to the user
7. maintenance: fixing bugs and adding new features

As the phases occur in a strict sequence, just as water never flows upwards, this model is like a waterfall.

This deductive or top-down description of the life cycle has its merits, especially with feedback.
The Waterfall Model
seven phases in sequence

The software life cycle model describes the software production in seven phases:

1. feasibility study: cost-benefit analysis
2. system requirements specification: what must it do?
3. system architecture design: modules and relations
4. implementation and verification of modules
5. system integration and verification
6. installation: deliver software to the user
7. maintenance: fixing bugs and adding new features

As the phases occur in a strict sequence, just as water never flows upwards, this model is like a waterfall.

This deductive or top-down description of the life cycle has its merits, especially with feedback.
The Waterfall Model
seven phases in sequence

The software life cycle model describes the software production in seven phases:

1. feasibility study: cost-benefit analysis
2. system requirements specification: what must it do?
3. system architecture design: modules and relations
4. implementation and verification of modules
5. system integration and verification
6. installation: deliver software to the user
7. maintenance: fixing bugs and adding new features

As the phases occur in a strict sequence, just as water never flows upwards, this model is like a waterfall.

This deductive or top-down description of the life cycle has its merits, especially with feedback.
The Waterfall Model
seven phases in sequence

The software life cycle model describes the software production in seven phases:

1. feasibility study: cost-benefit analysis
2. system requirements specification: what must it do?
3. system architecture design: modules and relations
4. implementation and verification of modules
5. system integration and verification
6. installation: deliver software to the user
7. maintenance: fixing bugs and adding new features

As the phases occur in a strict sequence, just as water never flows upwards, this model is like a waterfall.

This deductive or top-down description of the life cycle has its merits, especially with feedback.
The Waterfall Model

seven phases in sequence

The software life cycle model describes the software production in seven phases:

1. feasibility study: cost-benefit analysis
2. system requirements specification: what must it do?
3. system architecture design: modules and relations
4. implementation and verification of modules
5. system integration and verification
6. installation: deliver software to the user
7. maintenance: fixing bugs and adding new features

As the phases occur in a strict sequence, just as water never flows upwards, this model is like a waterfall.

This deductive or top-down description of the life cycle has its merits, especially with feedback.
The Waterfall Model
seven phases in sequence

The software life cycle model describes the software production in seven phases:

1. feasibility study: cost-benefit analysis
2. system requirements specification: what must it do?
3. system architecture design: modules and relations
4. implementation and verification of modules
5. system integration and verification
6. installation: deliver software to the user
7. maintenance: fixing bugs and adding new features

As the phases occur in a strict sequence, just as water never flows upwards, this model is like a waterfall.

This deductive or top-down description of the life cycle has its merits, especially with feedback.
The Waterfall Model
seven phases in sequence

The software life cycle model describes the software production in seven phases:

1. feasibility study: cost-benefit analysis
2. system requirements specification: what must it do?
3. system architecture design: modules and relations
4. implementation and verification of modules
5. system integration and verification
6. installation: deliver software to the user
7. maintenance: fixing bugs and adding new features

As the phases occur in a strict sequence, just as water never flows upwards, this model is like a waterfall.

This deductive or top-down description of the life cycle has its merits, especially with feedback.
The Waterfall Model
seven phases in sequence

The software life cycle model describes the software production in seven phases:

1. feasibility study: cost-benefit analysis
2. system requirements specification: what must it do?
3. system architecture design: modules and relations
4. implementation and verification of modules
5. system integration and verification
6. installation: deliver software to the user
7. maintenance: fixing bugs and adding new features

As the phases occur in a strict sequence, just as water never flows upwards, this model is like a waterfall.

This deductive or top-down description of the life cycle has its merits, especially with feedback.
The Waterfall Model
seven phases in sequence

The software life cycle model describes the software production in seven phases:

1. feasibility study: cost-benefit analysis
2. system requirements specification: what must it do?
3. system architecture design: modules and relations
4. implementation and verification of modules
5. system integration and verification
6. installation: deliver software to the user
7. maintenance: fixing bugs and adding new features

As the phases occur in a strict sequence, just as water never flows upwards, this model is like a waterfall.

This deductive or top-down description of the life cycle has its merits, especially with feedback.
The Spiral Model

Four sectors in the software development:

1. requirements analysis and specification
2. system architecture design
3. implementation
4. verification

Incremental development of software:

1. think of a product with limited scope
2. create the design of the first product
3. implement a prototype
4. test and deliver the prototype

Based on feedback, enlarge the specifications, revise the designs, build new prototypes, test and deliver again. As the cycles over the four sectors get larger, the complexity of the software grows.
The Spiral Model

Four sectors in the software development:

1. requirements analysis and specification
2. system architecture design
3. implementation
4. verification

Incremental development of software:

1. think of a product with limited scope
2. create the design of the first product
3. implement a prototype
4. test and deliver the prototype

Based on feedback, enlarge the specifications, revise the designs, build new prototypes, test and deliver again. As the cycles over the four sectors get larger, the complexity of the software grows.
The Spiral Model

Four sectors in the software development:

1. requirements analysis and specification
2. system architecture design
3. implementation
4. verification

Incremental development of software:

1. think of a product with limited scope
2. create the design of the first product
3. implement a prototype
4. test and deliver the prototype

Based on feedback, enlarge the specifications, revise the designs, build new prototypes, test and deliver again.

As the cycles over the four sectors get larger, the complexity of the software grows.
The Spiral Model

Four sectors in the software development:

1. requirements analysis and specification
2. system architecture design
3. implementation
4. verification

Incremental development of software:

1. think of a product with limited scope
2. create the design of the first product
3. implement a prototype
4. test and deliver the prototype

Based on feedback, enlarge the specifications, revise the designs, build new prototypes, test and deliver again. As the cycles over the four sectors get larger, the complexity of the software grows.
Summary + Assignments

Background reading for this lecture:

- chapter 7 in *Python Power!*
- section 7.2 in *Computer Science, an overview*

Assignments:

1. Enlarge the Q&A answer dictionary with a least six more questions about relevant CS concepts.
2. Modify the program so the user has the option to see the correct answer when wrong.
3. Design your own trivia quiz.
4. Decompose the quiz about the first 13 states (project 2) into modules, splitting the code into functions.