Developing an online learning platform and parameterized contents for Python Programming Language

2015 Summer

QuizPET Python Learning System is a Python Learning system providing parameterized programming exercises. Here is the link (http://columbus.exp.sis.pitt.edu/quizpet/). Meanwhile, we also created the contents (questions) for this platform organized in a hierarchical way, paying special attention to the relation among contents from a user modeling perspective.

Overview

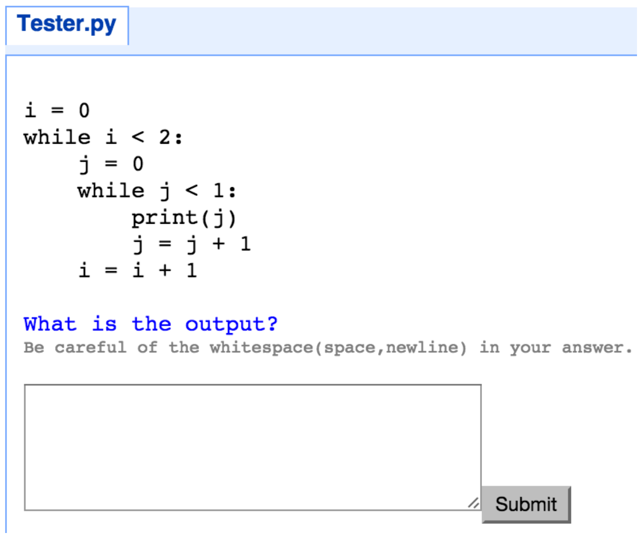
QuizPET has following characteristics:

- Each exercise is from a template with some parameters generated randomly each time a student attempts it.
- Each exercise evaluates student comprehension of a piece of code.
- Each exercise corresponds to a URL link. So it can be easily embedded in Mastery Grid or other systems.
- It reports students' responses to user model automatically, so that we can tracked students' correctness and response time.

Interface

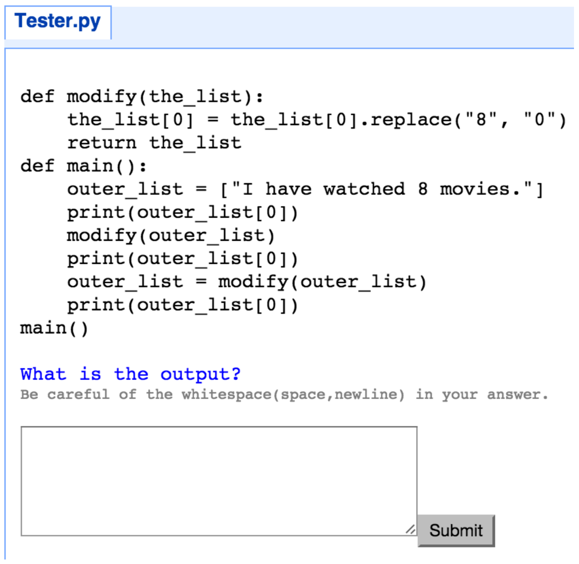
The interface is created involving several .jsp, .js, .htm, .css files following the same style and template as QuizJET (a learning platform for Java Programming language, in order to keep consistency with existed systems in the lab).

Here is a question for the topic *Loops*:   Here is a question for *List:*



```
i = 0
while i < 2:
    j = 0
    while j < 1:
        print(j)
        j = j + 1
    i = i + 1
```

What is the output?
Be careful of the whitespace(space,newline) in your answer.

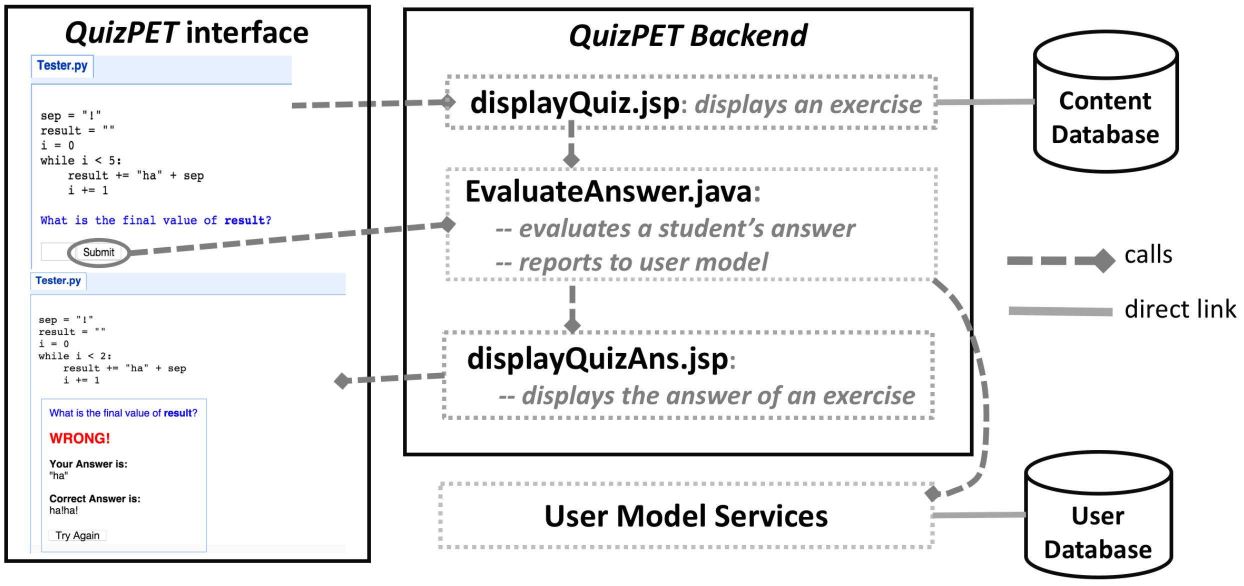


```
def modify(the_list):
    the_list[0] = the_list[0].replace("8", "0")
    return the_list
def main():
    outer_list = ["I have watched 8 movies."]
    print(outer_list[0])
    modify(outer_list)
    print(outer_list[0])
    outer_list = modify(outer_list)
    print(outer_list[0])
main()
```

What is the output?
Be careful of the whitespace(space,newline) in your answer.

Infrastructure

The overall infrastructure consists of the interface component, the backend Application service component, the backend user modeling service component, and the content and user database components. It is summarized as follows:



```
sep = "!"
result = ""
i = 0
while i < 5:
    result += "ha" + sep
    i += 1
```

What is the final value of **result**?

```
sep = "!"
result = ""
i = 0
while i < 2:
    result += "ha" + sep
    i += 1
```

What is the final value of **result**?

WRONG!

Your Answer is:
"ha"

Correct Answer is:
ha!ha!

Here is a more detailed explanation of the mechanism:

- Once a student clicks a question, the "displayQuiz.jsp" will query the content database and display the corresponding page;
- When a student clicks the "submit" button, the service "EvaluateAnswer.java" is called from "displayQuiz.jsp". In the backend, it

runs an interpreter on the corresponding Python code, gets the correct answer, compares it with the student's answer, and returns the judgment to "displayQuizAns.jsp". Meanwhile, it also reports to the user model by calling the related user model services.

- The user model service stores students' actions with detailed information including correctness, session, time (etc) into a centralized user database.
- "displayQuizAns.jsp" receives the call from "EvaluateAnswer.java", and then it displays the answer using the similar template as "displayQuiz.jsp".
- If a student clicks the "Try Again" button, it will be calling "displayQuiz.jsp" again, with a new instantiation of the variables of the same question template, and starts a new circle described as above.

Hierarchy of the Domain and Content Model

A domain model specifies the knowledge components (skills, concepts) required to be learned for the domain. A content model specifies the mapping between each piece of content (question, example, etc.) and the underlying required knowledge components. The design of QuizPET follows two level hierarchies as many of the tutoring systems. Each piece of content can be indexed in both topic level and concept level:

- Coarse-grained level, i.e., topics
- Fine-grained level, i.e., concept

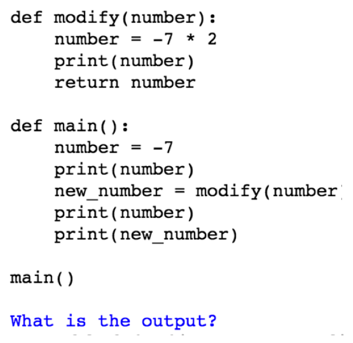
This helps student have a better sense of the syllabus, organization, and better manage their studying pace. Depending on the teachers' need, the same content can be re-organized into different topics.

Design Principles of the Contents

We design and create questions following the below principles:

- Questions are designed from easy to hard (within each topic)
- Questions in the same topic are targeting the most related set of concepts with different fined-grained learning objectives.
- Questions in latter topics have overlapping concepts with previous topics in order to help students to review, or deepen the knowledge of previous concepts by applying in different situations.

Here is an example of how fined-grained differentiation in learning objectives are addressed in the topic *Values and References*, different questions addressing mutable and immutable types are created:

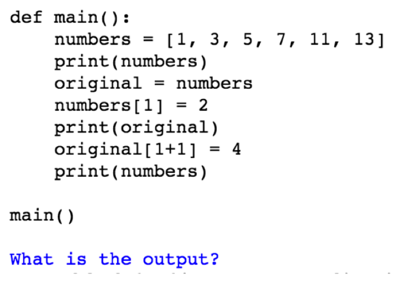


```
def modify(number):
    number = -7 * 2
    print(number)
    return number

def main():
    number = -7
    print(number)
    new_number = modify(number
    print(number)
    print(new_number)

main()
```

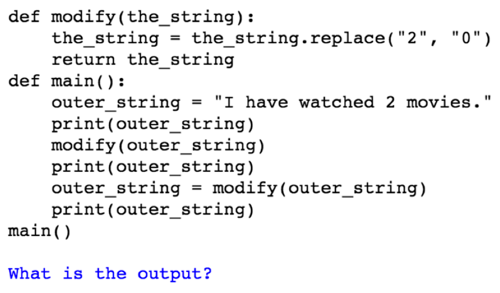
What is the output?



```
def main():
    numbers = [1, 3, 5, 7, 11, 13]
    print(numbers)
    original = numbers
    numbers[1] = 2
    print(original)
    original[1+1] = 4
    print(numbers)

main()
```

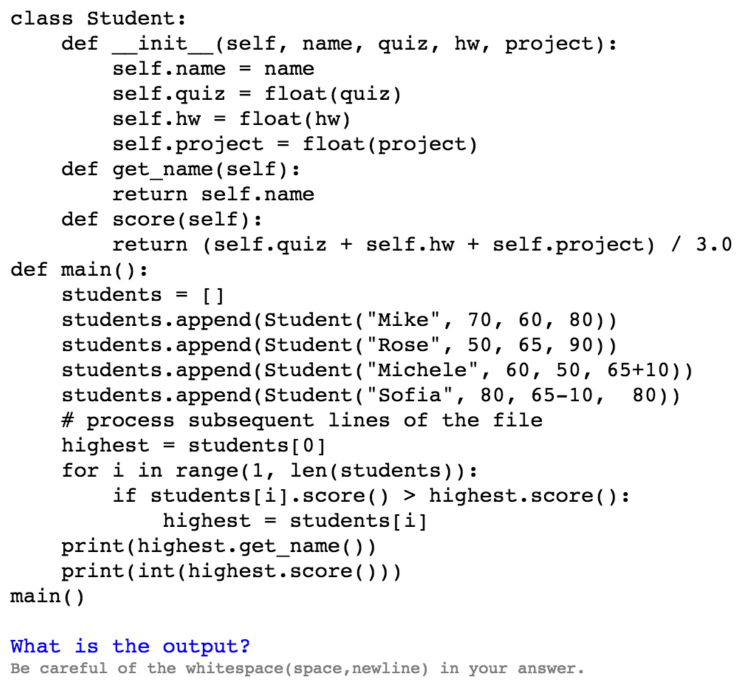
What is the output?



```
def modify(the_string):
    the_string = the_string.replace("2", "0")
    return the_string
def main():
    outer_string = "I have watched 2 movies."
    print(outer_string)
    modify(outer_string)
    print(outer_string)
    outer_string = modify(outer_string)
    print(outer_string)
main()
```

What is the output?

Here is an example of a question for the topic *Classes and Objects.* It requires concepts newly introduced within *Classes and Objects*, and it also covers three previous topics:



```
class Student:
    def __init__(self, name, quiz, hw, project):
        self.name = name
        self.quiz = float(quiz)
        self.hw = float(hw)
        self.project = float(project)
    def get_name(self):
        return self.name
    def score(self):
        return (self.quiz + self.hw + self.project) / 3.0
def main():
    students = []
    students.append(Student("Mike", 70, 60, 80))
    students.append(Student("Rose", 50, 65, 90))
    students.append(Student("Michele", 60, 50, 65+10))
    students.append(Student("Sofia", 80, 65-10,  80))
    # process subsequent lines of the file
    highest = students[0]
    for i in range(1, len(students)):
        if students[i].score() > highest.score():
            highest = students[i]
    print(highest.get_name())
    print(int(highest.score()))
main()
```

What is the output?
Be careful of the whitespace(space,newline) in your answer.

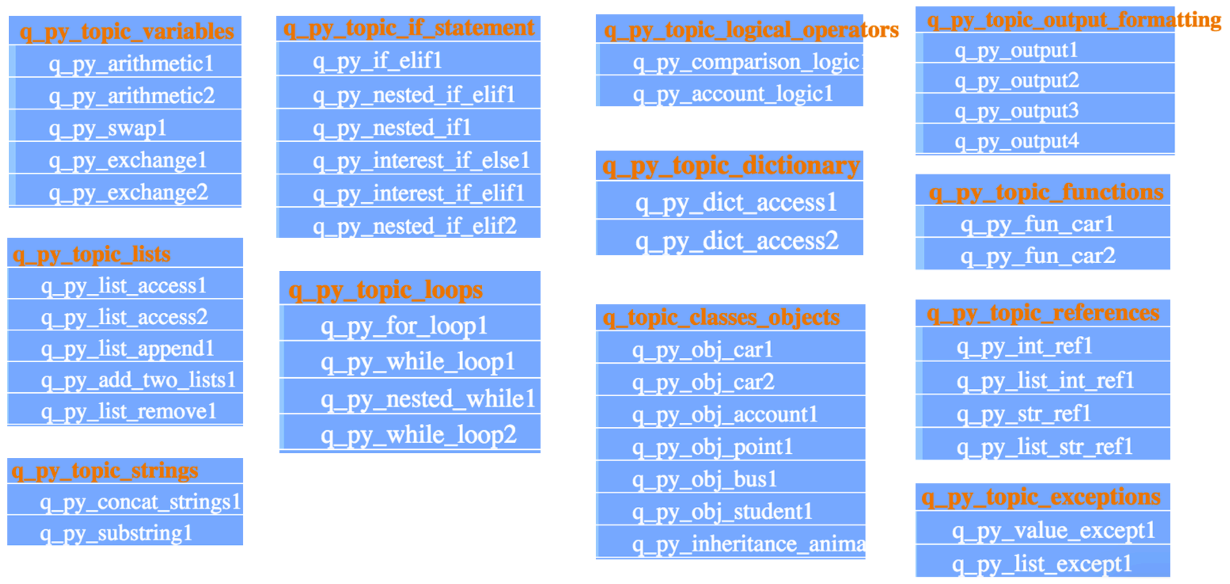
- Variables
- Comparisons
- If Statement
- Lists
- Loops
- Functions
- String

Contents

Currently, we have QuizPET exercises for 13 topics as follow:

- Variables, Comparisons, If Statement, Logical Operators, Loops, Output Formatting, Functions, Lists, Strings, Dictionary, Values and References, Exceptions, Classes and Objects

We have about 45 exercises in total. (Each exercise can be attempted different times with changes in parameters.) Each topic has 2 to 7 exercises. Here are the details of the content.



q_py_topic_variables
q_py_arithmetic1
q_py_arithmetic2
q_py_swap1
q_py_exchange1
q_py_exchange2

q_py_topic_lists
q_py_list_access1
q_py_list_access2
q_py_list_append1
q_py_add_two_lists1
q_py_list_remove1

q_py_topic_strings
q_py_concat_strings1
q_py_substring1

q_py_topic_if_statement
q_py_if_elif1
q_py_nested_if_elif1
q_py_nested_if1
q_py_interest_if_else1
q_py_interest_if_elif1
q_py_nested_if_elif2

q_py_topic_loops
q_py_for_loop1
q_py_while_loop1
q_py_nested_while1
q_py_while_loop2

q_py_topic_logical_operators
q_py_comparison_logic
q_py_account_logic1

q_py_topic_dictionary
q_py_dict_access1
q_py_dict_access2

q_topic_classes_objects
q_py_obj_car1
q_py_obj_car2
q_py_obj_account1
q_py_obj_point1
q_py_obj_bus1
q_py_obj_student1
q_py_inheritance_anima

q_py_topic_output_formatting
q_py_output1
q_py_output2
q_py_output3
q_py_output4

q_py_topic_functions
q_py_fun_car1
q_py_fun_car2

q_py_topic_references
q_py_int_ref1
q_py_list_int_ref1
q_py_str_ref1
q_py_list_str_ref1

q_py_topic_exceptions
q_py_value_except1
q_py_list_except1