Loops & List (cont)
Testing and Debugging

Intro2CS – week 3b
Reminder

• Solve exercises on your own.

• It is alright to talk to others about the exercise (but you need to specify who you talked to in the README)

• It is not alright to look at other people’s code or to get someone to write code for you.
def pyramid(height):
    for row in range(height):
        for _ in range(height-row-1):
            print(" ", end="")
        for _ in range(row*2+1):
            print("*", end="")
    print()

pyramid(10)
Nested lists

• We can place lists inside each other.

```python
records = [['Joe', 60], ['Jane', 95]]
print(records)
records.append(['Danny', 97])
records.append(['Tammy', 100])
print(records)

print(records[2])
print(records[2][0])
print(records[2][1])
```

```none
[['Joe', 60], ['Jane', 95]]
[['Joe', 60], ['Jane', 95], ['Danny', 97], ['Tammy', 100]]
['Danny', 97]
Danny
97
```
Bugs matter

• Therac-25 radiotherapy machine gave some patients 100x the amount of radiation it was supposed to.

• Mars orbiter fails (burns up in Mars’ atmosphere) because of a software bug
  – confusion about units (meters vs. feet)
How to find bugs in your code

While you write:

• Look at hints given by the IDE. Are some areas marked red?

• Write your program and test it as you go. Do not write all of it and only then test.
def run():
    records = []
    while True:
        action = input(INPUT_MESSAGE)
        if action == ACTION_SET_RECORD:
            pass  # TODO: Implement this!
        elif action == ACTION_REMOVE_RECORD:
            pass  # TODO: Implement this!
        elif action == ACTION_GET_AVERAGE:
            pass  # TODO: Implement this!
        elif action == ACTION_PRINT_STUDENTS:
            pass  # TODO: Implement this!
        elif action == ACTION_LOOKUP Grade:
            pass  # TODO: Implement this!
        elif action == ACTION_EXIT:
            return
        else:
            print("Unknown action selected. Please try again.")

run()
• Slowly fill in the code, and check it every once in a while.

```python
def run():
    records = []
    while True:
        action = input(INPUT_MESSAGE)
        if action == ACTION_SET_RECORD:
            name = get_name()
            grade = get_grade()
            if set_record(records, name, grade):
                print("Record changed."")
            else:
                print("Record added."")
        elif action == ACTION_REMOVE_RECORD:
            name = get_name()
            if remove_record(records, name):
                print("Done."")
            else:
                print("Record not found."")
        elif action == ACTION_GET_AVERAGE:
```

Create default implementations for these functions, and later fill them in:

```python
def get_name():
    return "Moe"
    # TODO: implement this!

def set_record(records, name, grade):
    return False
    # TODO: implement this!
Once the program is written

• Test edge cases!

• Enter strange input (negative grades, strings instead of numbers, etc.)
  – Every form of behavior should either be documented ("function assumes grades are non-negative numbers")
  – Or handled by the function

• Test every possible branch of an if statement

• Do not assume anything works well until you’ve checked it!
Can you spot the bug?

def compute_average_grade(records):
    grade_sum = 0
    for item in records:
        grade_sum += item[1]
    return grade_sum / len(records)
```python
def compute_average_grade(records):
    if len(records) == 0:
        return None
    grade_sum = 0
    for item in records:
        grade_sum += item[1]
    return grade_sum / len(records)
```
How to track down the bug

• Find a reproducible bug with a short sequence of inputs
• Bugs that crash the program: read the error message carefully.
• Actual bug could be in a different place
• When debugging, print the status of variables to see if they are what you expect them to be.
Writing “unit tests”

• Unit tests are pieces of code that check your main program

• Write the simplest possible tests

• Write independent tests whenever possible

• Write unit tests for every part of the program
Do we require unit tests in exercises?

• We will not ask you to write unit tests on all exercises.

• We may ask you to write some unit tests as part of an exercise.

AND / OR

• We will provide you with unit tests for an exercise that you can use to check your code.
Example

- Test this function:

```python
def set_record(records, name, grade):
    index = find_record_index(records, name)
    if index is None:
        records.append([name, grade])
        return False
    else:
        records[index][GRADE] = grade
        return True
```
def run_tests():
    test_set_record_1()
    test_set_record_2()
    test_set_record_3()
    test_remove_record_1()
    test_compute_average_grade_1()
    # add many more tests...

def print_pass_fail(boolean_test):
    if boolean_test:
        print("Pass")
    else:
        print("Fail")
White-box testing example

- Checks internal data structures for changes

```python
def test_set_record_1():
    print("Test set_record_1:", end="")
    my_records = []
    set_record(my_records,"name",100)
    print_pass_fail(my_records[0][0] == "name" and
                    my_records[0][1] == 100)

def test_set_record_2():
    print("Test set_record_2:", end="")
    my_records = [['name',100]]
    set_record(my_records,"name",50)
    print_pass_fail(my_records[0][0] == "name" and
                    my_records[0][1] == 50)
```
What else to test?

Be methodical. Check everything.

- Test the set record function when there are other records present
- Test its return value when record exists
- Test its return value when record does not exist
def remove_record(records, name):
    index = find_record_index(records, name)
    if index is None:
        return False
    else:
        del records[index]
    return True
Black box testing example

• Tests behavior of functions as a black box without looking inside the internal data structures.

```python
def test_remove_record_1():
    print("Test remove_record_1:", end="")
    my_records = []
    set_record(my_records, "name", 50)
    print_pass_fail(remove_record(my_records, "name"))
    # remove_record() should return True when record was found & removed.
```
• In many cases, testing code is 3x-5x longer than the code being checked!

• Why unit tests? When code is refactored, it is easy to run tests to see that everything still works.

• Makes it easy to re-test after every feature is added.
Many other ways to test

• Fuzz testing: throwing random input into your program to see that it still works

• Static analysis: automated tools used for seeing the code matches specification without actually running it

• ...