CIS 210

• Automated testing
• What’s wrong with this code?
• Midterm exam review
• Nested lists and aliasing
• Hexadecimal numbers
• Binary representation of characters

CIS 210 TESTING

That’s a lot of testing!

Test early, test often → automate testing to make it practical.

CIS 210 TESTING

Automated testing:
• supports comprehensive testing
• communicates comprehensive testing
• supports regression testing

So far – doctest.testmod() great as far as it goes, but for comprehensive testing
• docstrings too long
• no control over error report
• same tests may be useful for more than one system

⇒ LET’S WRITE OUR OWN TEST FUNCTIONS

CIS 210
def ctemp_to_ftemp(temp):
    """(number) -> float"
    return (ftemp = ctemp * 9/5 + 32)
    return ftemp

def test_ctemp_to_ftemp():
    """a very simple test function"""
    return (ctemp_to_ftemp(100) == 212.0 and
cmp_to_ftemp(0) == 32.0 and
cmp_to_ftemp(30) == 86.0 and
cmp_to_ftemp(21.1) == 69.98)

def test_ctemp_to_ftemp():
    """boolean"""
    if ctemp_to_ftemp(100) != 212.0:
        return False
    if ctemp_to_ftemp(0) != 32.0:
        return False
    if ctemp_to_ftemp(30) != 86.0:
        return False
    if ctemp_to_ftemp(21.1) != 69.98:
        return False
    return True

def test_ctemp_to_ftemp():
    """boolean"""
    return (ctemp_to_ftemp(100)) == 212.0 and
cmp_to_ftemp(0)) == 32.0 and
cmp_to_ftemp(30)) == 86.0 and
cmp_to_ftemp(21.1) == 69.98)
```python
def test_ctemp_to_ftemp():
    '''(number) - float'''
    print('Checking 100…', end='')
    result = ctemp_to_ftemp(100)
    if result == 212.0:
        print(' its value', result, 'is correct!')
    else:
        print(' Error: has wrong value', result, 'expected 212.0.')
    return False
if ctemp_to_ftemp(0) != 32.0:
    return False
if ctemp_to_ftemp(30) != 86.0:
    return False
return True
```

```python
def minutesToHours(minutes):
    def hoursToDays(hours):
        '''(number) - float'''
        convert input minutes to hours;
        return hours
        >>> minutesToHours(60)
        1.0
        >>> minutesToHours(90)
        1.5
        >>> minutesToHours(0)
        0.0
        ...
        hours = minutes / 60
        hours = round(hours, 2)
        print(hours)
        return None
        return not print
        return
```

```python
def daysToYears(days):
    def minutesToYears(m):
        '''(number) -> float'''
        convert input days to years;
        return years
        >>> daysToYears(365)
        1.0
        >>> daysToYears(100)
        0.27
        >>> daysToYears(0)
        0.0
        ...
        days = 365
        years = days / 365
        years = round(years, 2)
        return years
```

```python
def minutesToYears(m):
    >>> minutesToYears(525600)
    1.0
    >>> minutesToYears(394200)
    0.75
    ...
    minutesToHours(m)
    hoursToDays(h)
    daysToYears(d)
    return y
```

CIS 210

✓ Automated testing
- What’s wrong with this code?
- Binary representation of characters
- Nested lists and aliasing
- Midterm exam review
def daysToYears(days):
    """(number) -> float
    convert input days to years;
    input number m minutes is
    converted to equivalent
    number of years. return yrs.
    """

    years = days / 365
    years = round(years, 2)
    return years

def minutesToYears(m):
    """(int) -> float
    return years
    """

    years = m / (365 * 24)
    years = round(years, 2)
    return years

>>> daysToYears(365)
1.0
>>> daysToYears(100)
0.27
>>> daysToYears(0)
0.0

# remove line re-setting days

>>> li1 = [1, 2, [3, 4], [5, 6]]
>>> li2 = li1.copy()
>>> li1[0] = 'hi'
>>> li1
['hi', 2, [3, 4], [5, 6]]
>>> li2
[1, 2, [3, 4], [5, 6]]
>>> li1[2][0] = 999
>>> li1
['hi', 2, [999, 4], [5, 6]]
>>> li2
[1, 2, [999, 4], [5, 6]]

# ??!!

>>> li1 = [1, 2, [3, 4], [5, 6]]
>>> import copy
>>> li2 = li1.copy()
>>> li3 = copy.deepcopy(li1)
>>> li1[0] = 'hi'
>>> li1
['hi', 2, [3, 4], [5, 6]]
>>> li2
[1, 2, [999, True], [5, 6]]
>>> li3
['hi', 2, [999, True], [5, 6]]

# ??!!

CIS 210

✓ Automated testing
✓ What’s wrong with this code?
✓ Midterm exam review
• Nested lists and aliasing
• Hexadecimal numbers
• Binary representation of characters

CIS 210

binary representation of characters
UTF-8

>>> 'u03a9'
'Ω'
>>> 'u2877'
'‡'
Python/programming toolkit so far:

- numeric data types (int, float) and operations (e.g., +, *, **, round, abs)
- string data type and operations (e.g., +, len, count, find, format), formatted strings
- Python collections data types and operations – tuples, lists, dictionaries
- Boolean data type and operations (e.g., <, and)
- data type ”conversion” functions (e.g., str, int)
- NoneType (None)
- print/input expressions
- Python Standard Library – math, turtle, random modules; import
- assignment statement
- Python repetition – for, while
- Python conditionals – if
- variable assignment
- user-defined functions; function design; docstrings
- IDLE interactive development environment; help function

Programming/Computer Science Concepts

Computational Problem Solving: designing, implementing, checking, revising algorithms/programs.

Good programming style: function docstrings (type contract; description including parameters, returned value, and side effects if any; examples of function use), well-named variables, use of whitespace between operations and sections of code, judicious use of inline comments (why not what).

Python is a programming language and Python is an interpreter (program)
Python Shell is a REPL (read-evaluate-print loop)
Python primitive elements: Objects - value/attributes, type, memory location (id)
Combining primitive elements - Expressions - expressions evaluate to a value, overloaded operators; methods/functions/operators; short circuit evaluation of boolean expressions
Naming values: Variables/assignment - assignment statements are not expressions and do not return a value; namespaces – builtins and global (builtins and _._.); scope; dynamic typing; reference semantics.

Other language considerations – strong typing, mutable and immutable data types
Functions are an executable data type; what happens when a function is called:
- Activation record/block frame added to call stack for local namespace; return address
- Call-by-assignment parameter passing
- Functions always return a value (sometimes None)
- Functions sometimes have side effects

Functions as arguments
- Iterative algorithms; accumulator pattern; Monte Carlo algorithms; binary representation; encryption and decryption; data analysis

CIS 210 Learning Outcomes

- understand, develop, implement algorithms for computational problem solving;
- use structured design and testing methods to develop and implement programs;
- read, write, revise, document, test, and debug code;
- demonstrate robust mental models of data representation and code execution;
- demonstrate good understanding of a high level programming language;
- introduce and/or implement a sampling of classic computer science problem domains and algorithms.