CIS 210

Finishing / Starting

- More Python tools; map and filter patterns
- Software development environments
- Intro to user-defined classes
- Summing up

Final Exam Review – all labs (Q/A)

Recall: Finding an item in a sequence

def isIn(seq, t):
    """(seq: sequence, t: item) -> bool
    Search for item n in a sequence, seq. Return True if n is a
    member, else False.
    >>> isIn('hello', 'i')
    False
    >>> isIn((10, 20, 30, 40, 50, 60, 70, 80, 90), 80)
    True"
    for item in seq:
        if item == t:
            return True
    return False

Recall: Finding all occurrences of an item in a sequence

def findAll(seq, t):
    """(seq: sequence, t: item) -> list
    Search for item n in a sequence, seq. Return list of indices where
    t is found in seq.
    >>> findAll('hello', 'l')
    []
    >>> findAll((10, 20, 30, 40, 50, 60, 70, 80, 90), 80)
    [2, 4]"
    trackli = []
    for idx, item in enumerate(seq):
        if item == t:
            trackli.append(idx)
    return trackli

Recall: Finding an item in a sequence

def isIn(seq, t):
    """(seq: sequence, t: item) -> int
    Search for item n in a sequence, seq. Return index if n is a
    member, else return -1.
    >>> isIn('hello', 'i')
    -1
    >>> isIn((10, 20, 30, 40, 50, 60, 70, 80, 90), 80)
    7"
    trackli = []
    for idx in range(len(seq)):
        if seq[idx] == t:
            trackli.append(idx)
    return trackli

Recall: Finding an item in a sequence

def isIn(seq, t):
    """(seq: sequence, t: item) -> int
    Search for item n in a sequence, seq. Return index if n is a
    member, else return -1.
    >>> isIn('hello', 'i')
    -1
    >>> isIn((10, 20, 30, 40, 50, 60, 70, 80, 90), 80)
    7"
    for idx, item in enumerate(seq):
        if item == t:
            return idx

Recall: Finding all occurrences of an item in a sequence

def findAll(seq, t):
    """(seq: sequence, t: item) -> list
    Search for item n in a sequence, seq. Return list of indices where
    t is found in seq.
    >>> findAll('hello', 'l')
    []
    >>> findAll((10, 20, 30, 40, 50, 60, 70, 80, 90), 80)
    [2, 4]"
    trackli = []
    for idx, item in enumerate(seq):
        if item == t:
            trackli.append(idx)
    return trackli
Recall: Finding all occurrences of an item in a sequence

```python
def findAll(seq, t):
    '''(seq: sequence, t: item) - list
Search for item n in a sequence, seq. Return list of indices where
t is found in seq.
>>> findAll('hello', 'l')
[]
>>> findAll([10, 20, 80, 30, 40, 50, 60, 70, 80, 90], 80)
[2, 8]
'''
    trackli = []
    for idx, item in enumerate(seq):
        if item == t:
            trackli.append(idx)
    return [idx for idx, item in enumerate(seq) if item == t]
```

List Comprehensions

```python
def mode(alist):
    '''(list) -> list
    (declarative style syntax)
    Given:
    S = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    Generate:
    T is a list of x such that x is member of S and x is even
    ...
    modeli = []
    for k in countd:
        if countd[k] == maxct:
            modeli.append(k)
    better:
    modeli = [k for k in countd if countd[k] == maxct]
```

List Comprehensions

```python
return [idx for idx, item in enumerate(seq) if item == t]
```

list comprehensions create a list from a sequence
CIS 210
List Comprehensions

S = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
T is a list of x such that x is a member of S and x is even
{x | x ∈ S and x is even}  #mathematical

T = []  #procedural
for x in S:
    if isEven(x):
        T.append(x)

T = [x for x in S if isEven(x)]  #declarative (filter pattern)
[0, 2, 4, 6, 8, 10]

CIS 210
List Comprehensions

countd = {‘CIS’: 4, ‘EXPL’: 3}

# map pattern
doubleli = []
for k in countd:
    doubleli.append(countd[k] * 2)

>>> doubleli
??

CIS 210
List Comprehensions

List comprehensions are a concise way to create lists. The general syntax is:

[[<expression>]
for <item> in <sequence> ...
# other collection(s)
if <condition>]  # filtering (if needed)

Each item in the new list is the result of applying a given operation (<expression>)
to a value (<item>) from a sequence (<sequence>).

>>> [2 * i for i in [1, 2, 3]]  # ?? pattern
??
Finishing / Starting

More Python tools; map and filter patterns

- Programming environments
- Intro to user-defined classes
- Summing up

Final Exam Review – all labs (Q/A)

Algorithms → Programs

Programming Environments

- Editor for writing programs in a particular language and according to style guidelines
- Translator (interpreter/compiler, linker) for executing program (and reporting results)
- Other tools, e.g., for testing and debugging

Algorithms → Programs

Programming Environments

- Integrated Development Environment (IDE)
e.g., IDLE, PyCharm, Jupyter notebooks, Eclipse, ...

Individual tools also exist outside of IDEs:

- Text editor + Command line + Python debugging module (for example)

An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development.

[IDE] [PyCharm]
Once upon a time...

- switches
- cards
- teletype machines (command-line)
- video display terminals (command-line)
- Graphical User Interfaces (GUIs – point and click)
- command line emulators (Terminal/cmd.exe)

Integrated Development Environment (IDE) v. Command-line

Recall: p2c_sqrt_key_F20.py

```python
def main():
    '''Square root comparison program driver.'''
    sqrt_compare(25, 5)
    sqrt_compare(25, 10)
    ...
    return

main()
```
```
def import kfreeman
>>>
...+
This math
mysqrt
For
sysadmins
COMMAND LINE:
IDLE:
Integrated Development Environment (IDE) v. Command

sysadmins-air:F19-projects kfreeman$ python3.6 p2c_sqrt_key_F20.py
For 25 using 5 iterations:
mysqrt value is: 5.000023178253949
math lib sqrt value is: 5.0
This is a 0.0 percent error.
For 25 using 10 iterations:
mysqrt value is: 5.0
math lib sqrt value is: 5.0
This is a 0.0 percent error.
...

sysadmins-air:F19-projects kfreeman$
```

```
sysadmins-air:F20-projects kfreeman$ python3.6 p2c_sqrt_key_F20.py
For 25 using 5 iterations:
mysqrt value is: 5.000023178253949
math lib sqrt value is: 5.0
This is a 0.0 percent error.

>>> mysqrt
<function mysqrt at 0x100662e18>
>>> mysqrt(25, 10)
5.0
```

```
import sys

def main():
    """Use arguments supplied at command line""
    print(sys.argv)
    n = int(sys.argv[1])
    r = int(sys.argv[2])
    sqrt_compare(n, r)
    return

kfreeman$ python3.6 p2c_sqrt_key_F20.py 81 10
['p22_sqrt_key_F19.py', '81', '10']
For 81 using 10 iterations:
mysqrt value is: 9.0
math lib sqrt value is: 9.0
This is a 0.0 percent error.
```
Integrated Development Environment (IDE) v. Command-line

Writing – executing – testing and debugging code:

IDE: GUI, integrated, handy tools in one place, overhead to learn (and then learn again). Great for development and testing, especially for large projects.

command line: Shell (text only), not integrated, portable, powerful

built-in editors: e.g., emacs, vim, TextEdit, IDLE

---

Finishing / Starting

More Python tools; map and filter patterns

Programming environments
  - Intro to user-defined classes
  - Summing up

Final Exam Review – all labs (Q/A)

---

Python User-Defined Classes
A Short Intro

Recall: Python data types (objects):

type – range of values and operations
value(s)
id

---

A type, or class, is a template for objects.

Objects are instances of a class.

Every Python object is an instance of a class:

```python
>>> y = 'abc'
>>> type(y)
<class 'str'>
>>> y
'abc'
>>> id(y)
4298926768
>>> y.count('a')
4
```

---

```python
>>> y = 'abc'
>>> y = str(97403)
>>> type(y)
<class 'str'>
>>> y
'abc'
>>> id(y)
4298926768
>>> y.count('a')
4
```

---

```python
>>> y = str(97403)
>>> type(y)
<class 'str'>
>>> y
'97403'
>>> id(y)
4298926768
>>> y - 4
??
```

---

Every Python object is an instance of a class:

```python
>>> y = str(97403)  # y is an instance of class str
>>> type(y)  # returns class info
<class 'str'>
>>> y  # returns value info
'97403'
>>> id(y)  # returns memory location info
4298926768
>>> y - 4  # class/type restricts operations
TypeError
```

---

str is a constructor method for class str
str() instantiates a str object
y is an instance (object) of class type str with value and id

---
Recall: Python data types (objects):

- **type**: range of values and operations → attributes, object methods (operations), object descriptors
- **value(s)**
- **id**

**CIS 210**

Classes and Objects

Creating our own classes – Another way to extend the language as with functions: name, organize, encapsulate

now with data types: Object-oriented programming

**CIS 210**

Classes and Objects

Creating our own data types, or classes –

To create our own objects, or instances –

Example: Point

```python
>>> class Point:
    '''Represents a point in 2-d space.'''
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y

>>> p1 = Point()
>>> p1
<__main__.Point object at 0x15695d0>
```

**CIS 210**

Classes and Objects

special constructor method `__init__` initializes class names and values:

```python
class Point:
    '''Represents a point in 2-d space.'''
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y

>>> p1 = Point()
>>> p2 = Point(3, 4)
```
CIS 210 Classes and Objects

Add methods to get attribute values:

```python
class Point(object):
    '''Represents a point in 2-d space.'''
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y
    def getx(self):
        return self.x
def setx(self, newx):
    return self.x = newx
    def gety(self):
        return self.y
def sety(self, newy):
    return self.y = newy
```
Classes and Objects

__str__ is a special method used to print the object using print

class Point(object):
   '''Represents a point in 2-d space.'''
   def __str__(self):
      return f'***{{self.x:0.0f}}, {{self.y:0.0f}}***'

>>> p4 = Point(0, 100)  # execute __init__ method
>>> p4.setx(99)  # execute setx method
>>> print(p4)  # execute __str__ method
***99, 100***

Besides __init__, other special ('magic') methods: __str__ and __add__, for example

__add__ is a special method used by the + operator
its behavior varies per object type (overloaded operators)

def __add__(self, val):
   '''type-based dispatch'''
   if isinstance(val, Point):
      self.addpoint(val)
   else:
      self.addint(val)

def addint(p, i):
   '''Add i to x and y values of p.'''
   p.x += i
   p.y += i
   return

def addpoint(p1, p2):
   '''Add x and y values of p2 to p1.'''
   p1.x += p2.x
   p1.y += p2.y
   return
CIS 210 Classes and Objects

```python
def __add__(self, val):
    '''type-based dispatch'''
    if isinstance(val, Point):
        return self.addpoint(val)
    else:
        self.addint(val)

def addint(p, i):
    '''Add i to x and y values of p.'''
    p.x += i
    p.y += i
    return p

def addpoint(p1, p2):
    '''add p1 and p2 x, y vals to create new point.'''
    return Point(p1.x + p2.x, p1.y + p2.y)
```

CIS 210 / Welcome

**Computational Problem Solving**

- Understand/revise/refactor/create/implement algorithms and code
- Expand/improve software development skills
- Demonstrate robust mental models of data representation and code execution
- Gain familiarity with a sampling of classic computer science problem domains/algorithms.

**Computational Problem Solving**

is an approach to problem solving that is inspired and constrained by the possibilities and limitations of computers and computing.

**Finishing / Starting**

- More Python tools; map and filter patterns
- Programming environments
- Intro to user-defined classes
  - Summing up
    - Final Exam Review – all labs (Q/A)

**What you can expect from CIS 210**

**CIS 210 / Welcome**
• Understand/revise/refactor/create/implement algorithms and code

• Expand/improve software development skills
  – use structured design and testing methods to develop and implement programs
  – good understanding of a high level programming language

programming: coding (bottom up)
  review Python toolkit
  write Python code (use pseudocode to start)

programming: testing and debugging
  test code using examples of use (automated testing)
  revise and retest as needed

  • cases for basic, edge, for various expected arguments and results (equivalence classes)
  • syntax, runtime, logical, documentation
  • assert and try/except

demonstrate good understanding of a high level programming language
Python toolkit (wk 10)
  numeric data types (int, float) and operations (e.g., +, *, //, pow, round, abs)
  string data type and operations (e.g., +, *, in, count, find), formatted strings
  boolean data type and operations (e.g., True, False)
  data type coercion functions, e.g., int, str
  Python collections data types and operations – tuples, lists, dictionaries; list comprehensions
  Python file and file processing
  very short intro to user-defined objects (classes)
  for loop
  if/else
  while
  Python conditionals (selection) – if
  user-defined functions; def, parameter list, docstring, return (values)
  IDLE interactive development environment
  very short intro to other programming environments (PyCharm IDE and command-line programming)
  Python introspection – help, type, dir
  Python Standard library – math, random, turtle, datetime, csv, modules, import
  short intro to other data structures (e.g., stacks, queues, linked lists)

Python is a language (virtual machine) AND
Python is a program (translator/interpreter)

CIS 210 / Welcome
demonstrate robust mental models of data representation and code execution

Python is a language (virtual machine) AND
Python is a program (translator/interpreter)
CIS 210

Finishing / Starting

• More Python tools; map and filter patterns
• Programming environments
• Intro to user-defined classes
• Summing up

Final Exam Review – all labs (Q/A)
A Structured Approach to Computational Problem Solving

- review the project specification thoroughly
- write examples of expected results for specified inputs - review spec, if needed
- develop, review, and/or revise a problem-solving approach, using natural language, algorithm, pseudocode (not Python code)
- check algorithms using your examples - revise algorithm, review spec, if needed

Starting with the lowest level function -
- write the function header
- write the function docstring - type contract
- write the function docstring - brief description
- write the function docstring - examples of use (use ones developed earlier)
- write the return statement

- using tools from the Python toolkit, start writing the body of the function
- test often, revise as needed
- test using examples in the docstring, and then project spec, and then others

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, review, 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.