COMPSCI 105 S1 2017
Principles of Computer Science

Revision

Today's lecture

- Topics
- Term Test Overview
- How to prepare for your test
- Revision/Exercises

Test Information

- The test is worth 15% of your final mark.
- Date: **Monday 3rd April 2017**
- Time: 6:15pm - 7:15pm (Please arrive by 6pm as you will be given 5 minutes' reading time.)
- Read the instructions on filling out a Teleform sheet before you go to the test
  - How to fill out a Teleform sheet
- Please bring your Student Id card, a pencil and an eraser

Room Allocations

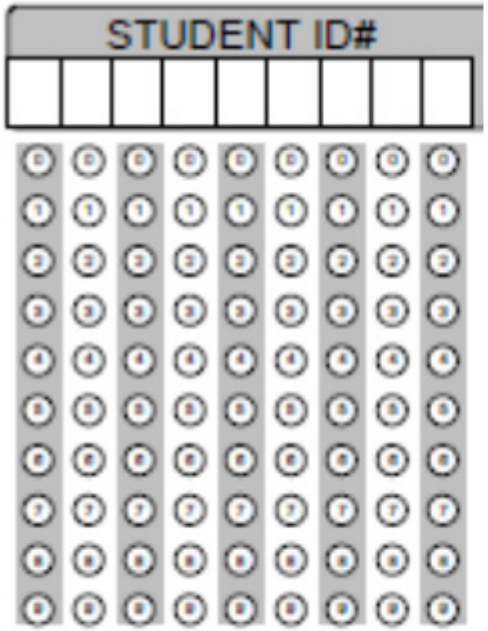
- You have been allocated one of these rooms in which to sit the test, based on your surname. Please attend the test in the room corresponding to your surname:
  - PLT1/303-G20: Surname A – K
  - HSB1/201N-346: Surname L – Z

Test Information

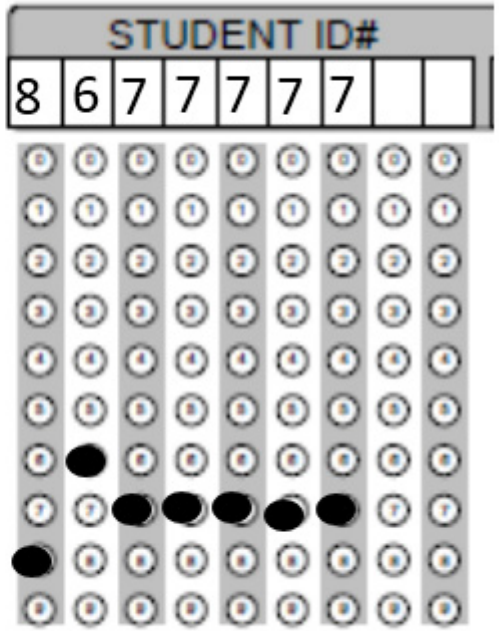
- Time Allowed: 1 hour
- Closed book, no calculator
- Calculators are NOT permitted.
- Please notify Angela if you have a test clash
- You must answer all questions in this exam.
  - Answer Section A (Multiple choice questions) on the Teleform answer sheet provided.
  - Answer Section B in the space provided in this booklet.
- All material from the lectures, assignments and labs is relevant unless specifically
- Both questions and answer choices may contain Python code.

Teleforms

- Fill in your Student ID Number in the STUDENT ID# section.
- Also fill in one column for each digit of your Student ID Number.
- The example below shows a Student ID Number of 8677777 (with 7 digits)



STUDENT ID# Section *Blank*



STUDENT ID# Section *Example*

Test Overview

- Section A: Multiple choice questions
  - 25 questions, total = 34 marks
  - Topics:
    - Python Programming - functions (3 questions)
    - Lists (3 questions)
    - List comprehensions (3 questions)
    - Equality, references and mutability (2 questions)
    - Classes (5 questions)
    - Exceptions (4 questions)
    - JSON (1 question)
    - Complexity/Algorithm Analysis (4 questions)

Test Overview

- Section B: Short answer questions
  - 2 questions, total = 16 marks
  - Question 26: 10 marks (Write a custom class)
  - Question 27: 6 marks
    - a) Write a function…
    - b) Write a function…

Why MCQ

- Because multiple choice exams contain many questions, they require students to be familiar with a much broader range of material than open answer exams do
- Multiple choice exams also usually expect students to have a greater familiarity with details
- Lower risk for students -
  since there are more questions, misunderstanding/misreading a question has less severe results
- Faster to mark (results are out earlier)

Test Technique

- Get enough sleep before the test
- Plan your exam – identify easy questions and do them first
  - this boosts your confidence and avoids that you loose easy points because you run out of time
- When reading the question cover up the answer choices
  - anticipate answer before seeing the possible answers
  - If you see expected answer circle it, but check out other answers if one of them is better
  - If you can't answer a question (say, 1-2 min) come back to it later
- If you run out of time at the end, do informed guesses
- Don't panic :
  - remember that everyone has to answer the same questions...

Summary & Exercises

- Python Programming
- Sequences
- Lists
- List comprehensions
- Equality, references and mutability
- Classes
- Exceptions
- JSON
- Complexity/Algorithm Analysis

Python Programming

- Expression
  - Arithmetic operators
  - Relational operators >, >=, <, <=, ==
  - Boolean operators and, or, not
- Conditionals
- Functions: name, arguments, return value
  - Arguments: Default values
- Loops
  - for, while
  - The else clause
  - Break, continue
  - range

Python Operator Precedence

Operator	Description
`()`	Parentheses (grouping)
`f``(args...)`	Function call
`x``[index:index]`	Slicing
`x``[index]`	Subscription
`x.attribute`	Attribute reference
`**`	Exponentiation
`~`*`x`*	Bitwise not
`+`*`x`*`, -`*`x`*	Positive, negative
`*, /, %`	Multiplication, division, remainder
`+, -`	Addition, subtraction
`<<, >>`	Bitwise shifts
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`in, not in, is, is not, <, <=, >, >=, <>, !=, ==`	Comparisons, membership, identity
`not` *`x`*	Boolean NOT
`and`	Boolean AND
`or`	Boolean OR
`lambda`	Lambda expression

Sequences

- Mutable & Immutable
  - An immutable object is an object whose state cannot be modified after it is created
    - Immutable: Tuple, String
    - Mutable: List
- Operations:

Operation Name	Operator	Explanation
indexing	`[ ]`	Access an element of a sequence
concatenation	`+`	Combine sequences together
repetition	`*`	Concatenate a repeated number of times
membership	`in`	Ask whether an item is in a sequence
length	`len`	Ask the number of items in the sequence
slicing	`[ : ]`	Extract a part of a sequence

```
>>> name = 'Andrew'
>>> name[::-1]
```

'werdnA'

  - Slicing:
    - If the step size is **negative**, it starts at the end and steps backward towards the start.

Lists & List Comprehension

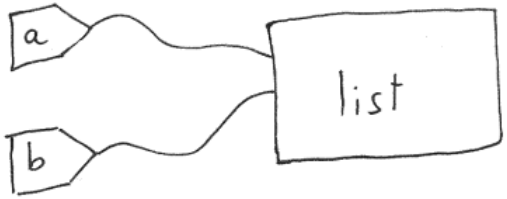
- Methods:
  - append, insert, remove, extend, reverse, sort…
- Operators
  - +, += etc
- List Comprehension

```
[expression for variable in sequence if condition]
```

Try MCQ exercises from CodeRunner

Reference: == Vs is

- ==
  - Checks if the value of two operands are equal or not, if yes then condition becomes true
- is
  - Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.



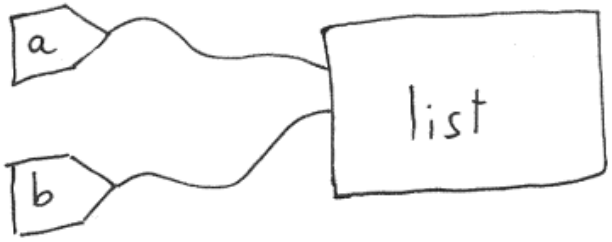
Lists

- Part 1:

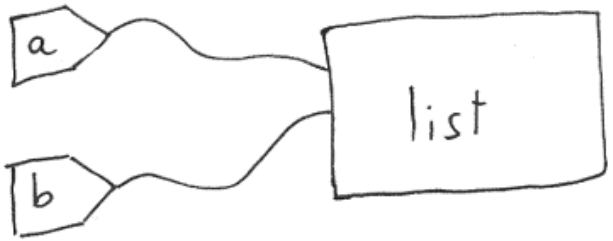
```
def q1_1():
    list_a = [1, 2, 3]
    list_b = list_a
    print(list_a == list_b, list_a is list_b)
```

- Part 2:

```
def q1_2():
    list_a = [1, 2, 3]
    list_b = list_a
    list_b.append(4)
    print(list_a)
    print(list_b)
    print(list_a == list_b, list_a is list_b)
```



Lists

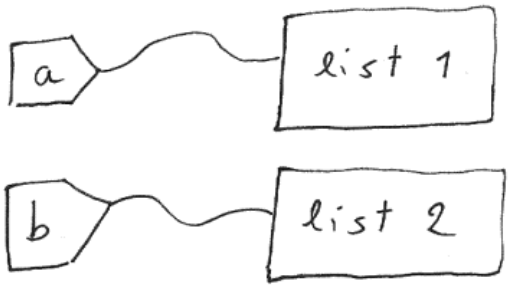


- Part 3:

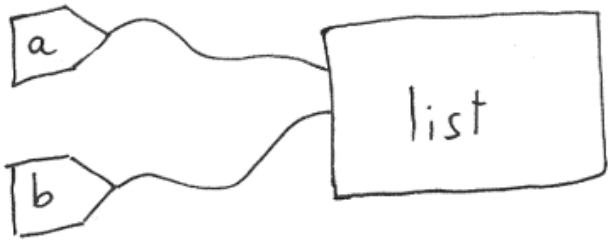
```
def q1_3():
    list_a = [1, 2, 3]
    list_b = list_a
```

Create a new list

```
    list_b = list_b + [4]
    print(list_a)
    print(list_b)
    print(list_a == list_b, list_a is list_b)
```



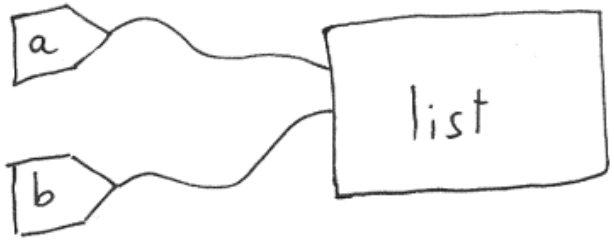
Lists



- Part 4:

```
def q1_4():
    list_a = [1, 2, 3]
    list_b = list_a
```

```
    list_a += [4]
    print(list_a)
    print(list_b)
    print(list_a == list_b, list_a is list_b)
```



The MobilePhone Class

Jonathan has an Apple iPhone 4
Alastair has a Sumsung Galaxy

- Complete the MobilePhone class so that when the following code fragment is run, it produces the output as above.

```
jonathansPhone = MobilePhone("Apple", "iPhone 4")
alastairsPhone =  MobilePhone("Sumsung", "Galaxy")
print("Jonathan has an " + str(jonathansPhone))
print("Alastair has a " + str(alastairsPhone)
```

The MobilePhone Class

- Complete the constructor, __repr__, __str__ method

```
class MobilePhone:

    def __init__(self, brand, model):




    def __repr__(self):
        return 'MobilePhone({0}, {1})'.format(              ,          )

    def __str__(self):
        return
```

Point class (add, radd, iadd)

```
p = Point(1,2)
q = Point(3,4)
```

	No __add__	with __add__
r=p+q print(r)	TypeError: unsupported … +: 'Point' and 'Point'	(4, 6)

	with __add__	with __add__ + isinstance(other, Point)
r=p+2 print(r)	AttributeError:…no attribute 'x' *(case 3)*	(3, 4) *(case 4)*

	with __add__ + isinstance(other, Point)	with __radd__
r=2+p print(r)	TypeError: unsupported … +:'int' and 'Point' *(case 5)*	(3, 4) *(case 6)*

	with __add__ + isinstance(other, Point)	with __iadd__
r = p p += q print(r, p, r is p)	(1, 2) (4, 6) False *(case 7)*	(4, 6) (4, 6) True #make changes in place *(case 8)*

2015 S1 1-8 : Square class

```
class Square:
    def __init__(self, s):
        self.side = s
    def perimeter(self):
        return 4*self.side
    def area(self):
        return self.side * self.side
    def scale(self, factor):
        self.side *= factor
```

2015 S1 1-8 : Square class

```
    def __le__(self, other):
        if not isinstance(other, Square):
            return False
        return self.area() <= other.area()
    def __ne__(self, other):
        if not isinstance(other, Square):
            return False
        return self.area() != other.area()
    def __eq__(self, other):
        if not isinstance(other, Square):
            return False
        return self.area() == other.area()
```

Q1-2

```
r = Square(5)
s = Square(10)
```

Which of the following code fragment will have **False** as output?

(a) print(r == s)
(b) print(r.area())
(c) print(r.perimeter())
(d) print(r <= s)
(e) None of the above

Different area()
25 Vs 100

```
r = Square(5)
s = Square(10)
t = Square(2)
u = Square(4)
```

Which of the following code fragment will cause a TypeError exception?

(a) print(s.area())
(b) print(u < s)
(c) print(t.perimeter())
(d) print(r == u)

__lt__ method?

Q3-5

```
r = Square(5)
s = Square(10)
r.scale(2)
print(r != s, r.area())
```

r.Side = 10
s.Side = 10
Same area
False 100

```
r = Square(5)
s = Square(10)
t = Square(4)
u = Square(6)
print(r != s, s <= r, u <= t)
```

r.area() = 25
s.area() = 100
t.area() = 16
u.area() = 36
True False False

```
r = Square(5)
s = Square(10)
u = r
print(r != s, r is not s, s is u, s != u)
```

r.area() = 25
s.area() = 100
u and r = same memory location

r != s (25 Vs 100) -> True
r is not s (two objects) -> True
s is u (two objects) -> False
S != u (area: 100 Vs 25) True

Q6-8

```
r = Square(5)
s = Square(10)
a = [r, s]
b = a.copy()
print(a != b, a is b, a[0] != b[0], a[0] is not b[1])
```

a, b (two objects)
a[0] and b[0] refer to the same location

a != b (value of two lists are the same): False
a is b (two objects): False
a[0] != b[0]: (same object, content must be the same: False
a[0] is not b[1] (two objects): True

```
from copy import deepcopy
r = Square(10)
s = Square(5)
a = [r, s]
b = deepcopy(a)
print(a == b, a is not b, a[0] != b[1], a[0] is b[0])
```

A & b (two objects)
a[0] &b[0] (two objects)

a == b (value of two lists are the same): True
a is not b: two objects: True
a[0] != b[1] : different area() : True
a[0] is b[0]: (two objects): False

```
r = Square(5)
print(r != 5)
```

__ne__
isinstance(other, Square) ->
method return false

Exceptions

- What is the output of the following code fragment?

```
result = ''
try:
    num = 100 / 0
    result += 'a'
except ZeroDivisionError:
    result += 'b'
except:
    result += 'c'
finally:
    result += 'd'
print(result)
```

Exceptions

- Given the following code fragment…

```
result = ''
try:
        num = ...
        result += 'a'
        try:
                num = 200 / 0
                result += 'b'
        except ValueError:
                result += 'c'
        except:
                result += 'd'
        finally:
                result += 'e'
except ZeroDivisionError:
        result += 'f'
finally:
        result += 'g'
print(result)
```

- What is the output if …

```
num = int('Hello')
```

Traceback (most recent call last):…..
…
ValueError: invalid literal for int() with base 10: 'Hello'

Exceptions

- Given the following code fragment…

```
result = ''
try:
        num = ...
        result += 'a'
        try:
                num = 200 / 0
                result += 'b'
        except ValueError:
                result += 'c'
        except:
                result += 'd'
        finally:
                result += 'e'
except ZeroDivisionError:
        result += 'f'
finally:
        result += 'g'
print(result)
```

- What is the output if …

```
num = 100
```

Exceptions

- Given the following code fragment…

```
result = ''
try:
        num = ...
        result += 'a'
        try:
                num = 200 / 0
                result += 'b'
        except ValueError:
                result += 'c'
        except:
                result += 'd'
        finally:
                result += 'e'
except ZeroDivisionError:
        result += 'f'
finally:
        result += 'g'
print(result)
```

- What is the output if …

```
num = 100 / 0
```

Complexity

- Big O for an algorithm
- Comparison of Growth Rate:

$O(1) < O(\log_2 n) < O(n) < O(n * \log_2 n) < O(n^2) < O(n^3) < O(2^n)$

- Calculating Big-O
  - Straight line code
  - Loops
  - Nested Loops
  - Consecutive Statements
  - If-then-else statement
  - Logarithmic
- Best-case, Worst-case, Average-case analysis

Complexity

- What is the Big-O complexity of the following function?

```
def q6(my_list):
    n = len(my_list)
    result = 100
    i = n -3
    while i > 0:
        result = result // my_list[i]
        i = i // 2
    return result
```

Complexity

- What is the Big-O complexity of the following function?

```
def q7(n):
    amount = 0
    i = n
    while i > 1:
        for j in range(1,5):
            amount = amount + j * 3
        i = i // 2
    return amount
```

O(n)

- If we have an algorithm that is O(n) and it will run for 10 seconds for a problem size 1000
- Now if we have the problem size 2000, what is the approximate run time in seconds?

- This means that if we were to make a graph showing how the number of inputs, n, relates to the amount of time required to perform the task, that we would expect the graph to be linear, if 1000 inputs take 10 seconds, then 2000 inputs takes 20 seconds

O(n^2)

- O(n^2) would be read as "Big-oh of n squared"

- If 1000 inputs takes 30 seconds, then 2000 inputs takes 2 minutes.
- How did we get that? Because n^2 where we are doubling n
  - => (2n)^2 => (2^2)(n^2) => 4(n^2)
  - And we know that n^2 is 30 seconds, so 4(n^2) must be 4 * 30 seconds which is 2 minutes.