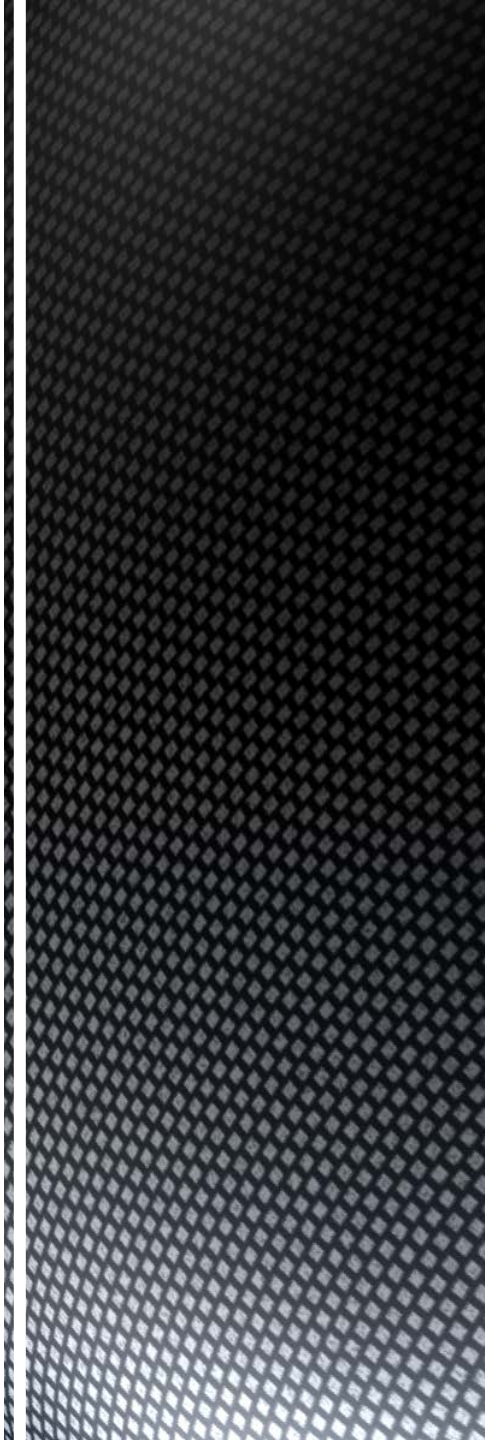
COMPSCI 107
Computer Science Fundamentals

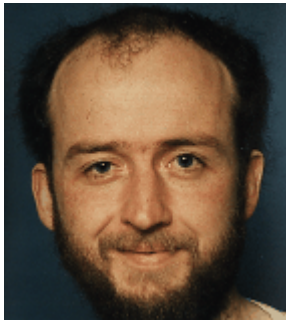
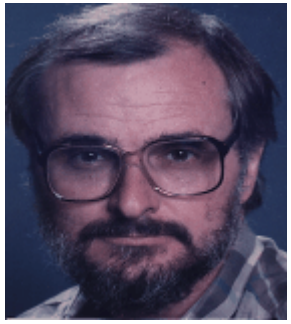
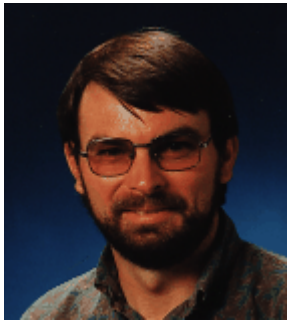
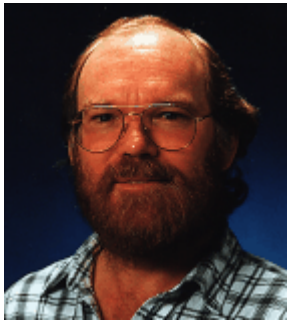
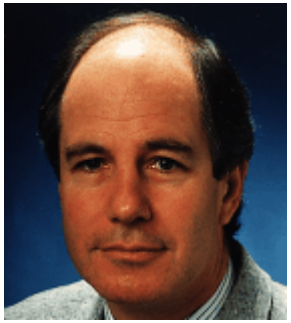
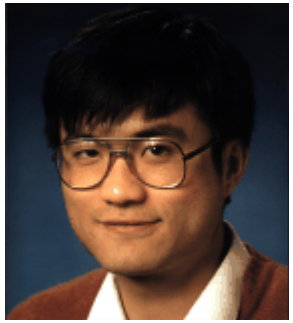
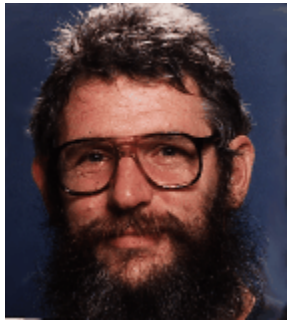
Introduction



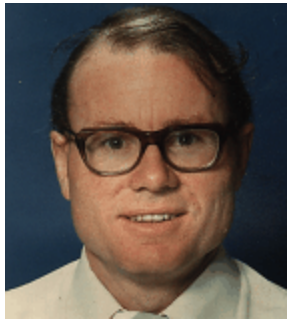
Waiheke Island

Philosophy

Computer Science





Research Interests

- ACM Special Interest Group in Computer Science Education

My kids

Research on lectures

- An efficient way to teach – not an effective way to learn

Research on learning

What is effective?

- Actively doing something (bring laptop to class if possible)
- Critically evaluating yourselves
- Learning from peers

Assessment

- Laboratories every week (starting week 2) …… 25%
  - Exercises during labs
  - Exercises after labs(homework)
- Mid-semester Test …… 15%
  - 1st May 205, in class, during normal lecture time
- Final Exam …….. 60%
  - Date to be announced

Feedback

- Computer feedback
  - Errors, testing, debugging
- Automated marking
  - CodeRunner – automated testing
- Laboratory feedback
  - Demonstrators
  - Group code review

Resources

- Lectures
  - Overheads and recordings
- Forum
  - Question and answers – peers, tutors and lecturers
- Textbook
  - Problem Solving with Algorithms and Data Structures
  - Online, free, open source
- Additional resources
  - Python.org
  - PythonTutor.com

COMPSCI 107 Curriculum overview

Making informed choices about programming

- Building mental models of data storage and control flow
- Understanding the trade-offs

Focus on ways of storing and manipulating data

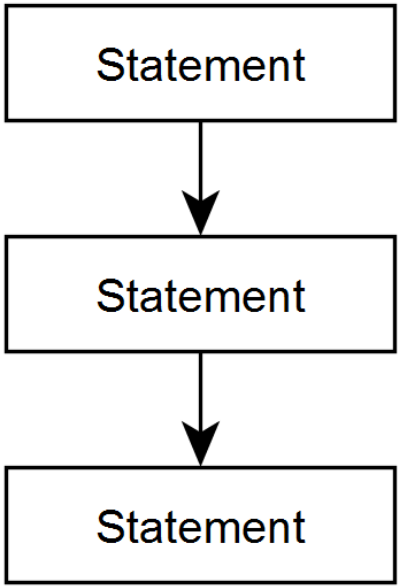
- Different ways of structuring data storage
- Efficiency
- Searching
- Sorting

Some new programming ideas

- Recursion
- Exceptions

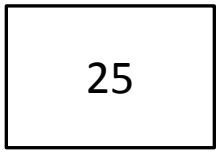
Programs

- Programs consist of one or more instructions
- Instructions are executed in a sequence



Variables

- A simple storage box
  - name of the box is the identifier
  - stores only one thing at a time



- Information in a variable has a **type**
  - boolean, integer, float, string
  - you can perform different operations on different types of data

- Values are stored in variables using an assignment statement

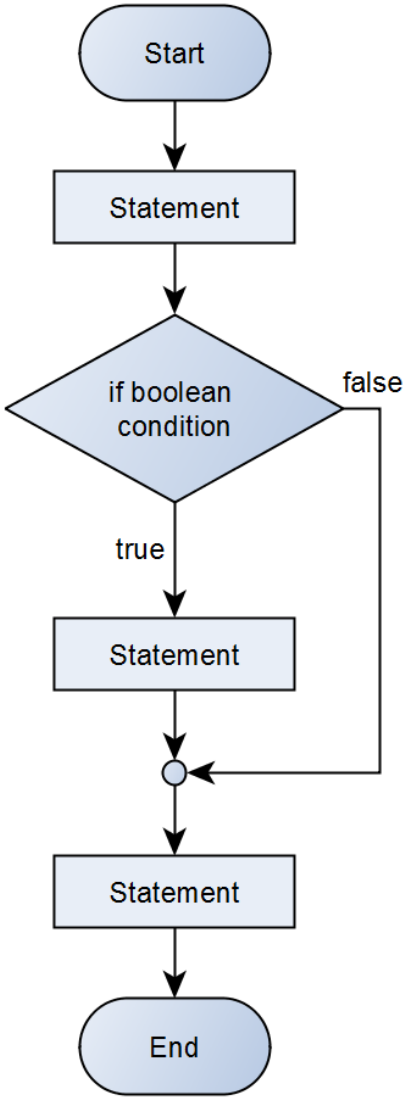
```
name = "Andrew"
age = 25
```

Selection

- Conditional statements
  - Change the flow of control
  - Conditions must evaluate to true or false (boolean)
  - Execute additional statements if condition is true

- Used to make decisions

```
discount = 0.0
age = int(input("Enter your age: "))

if age >= 65:
    discount = 0.10
    print("You get a discount of {:.0%}".format(discount))

price = 35 * (1.0 - discount)
print("Your tickets cost: ${:.2f}".format(price))
```

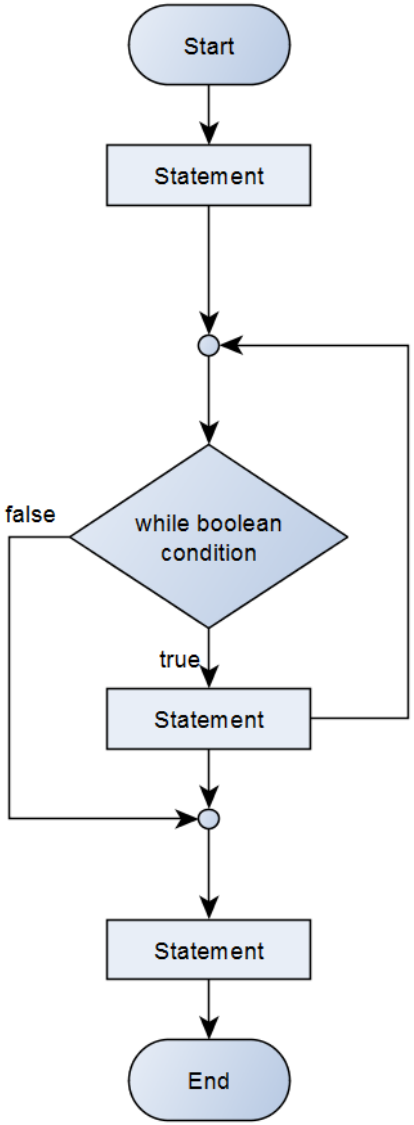


Iteration

- Loops
  - Change the flow of control
  - Execute a set of statements multiple times
  - Branch when the condition is false

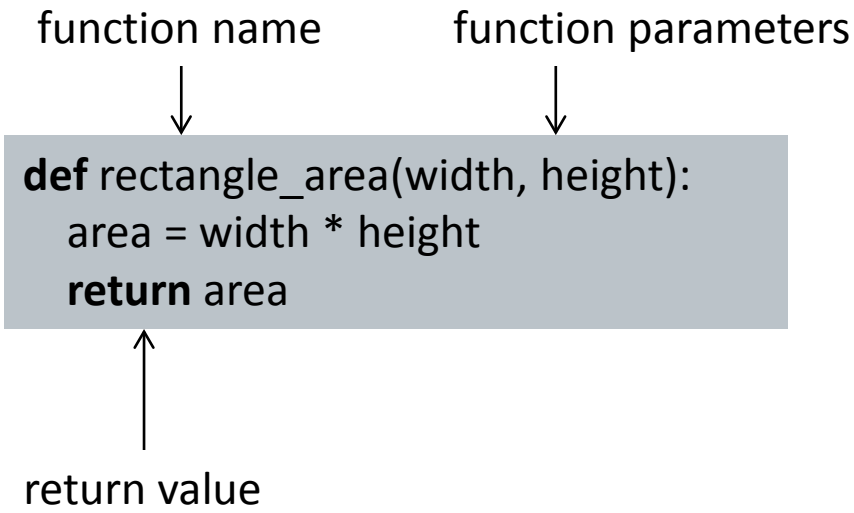
- Similar to a conditional
  - but repeats the statements
  - Python hides some of the details

```
tables = int(input("Enter the times tables: "))
print("===============")

for i in range(1, 11):
    print("{} x {} = {}".format(i, tables, i * tables))

print("===============")
```



Functions

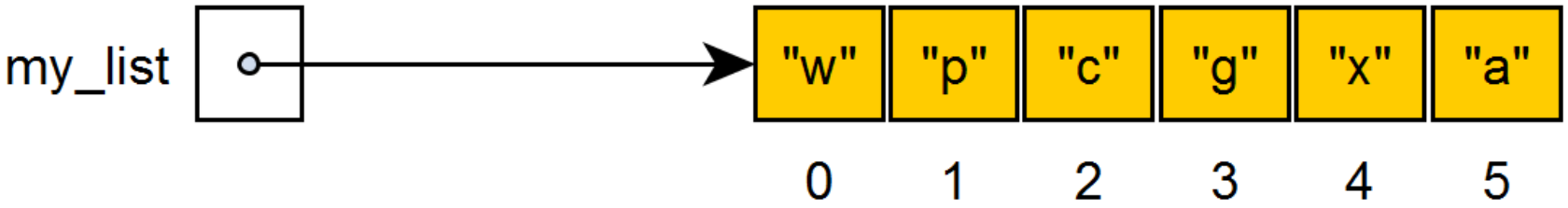
- Functions (procedures, methods, subroutines) are a way to group statements together as a unit
  - An identifier is used to label the function (function name)
  - Parameters make the code more general
  - Code can return a value



Structured data - arrays

- Arrays
  - Python hides these, but they are fundamentally important
  - Single variable name refers to a sequence of variables
  - Integer values used as an "index" to specify which variable in the sequence is required
  - Position of the information in memory can be calculated using formula:

    location = location_of_position_0 + index * memory_size_of_each_element




```
my_list = ["w", "p", "c", "g", "x", "a"]
print(my_list[0])
print(my_list[3])
```

Encapsulating data

- Object oriented programming
  - Defining a class (type)
  - Group data and code operating on that data

Modular, reusable

- Separate interface from implementation

```
class coordinates:
   #Note,  coordinates are (x, y) integer values on cartesian plane

   def __init__(self, x, y):
      self.x = x
      self.y = y

   def __repr__(self):
      return "({}, {})".format(self.x, self.y)
```

Dealing with incorrect data

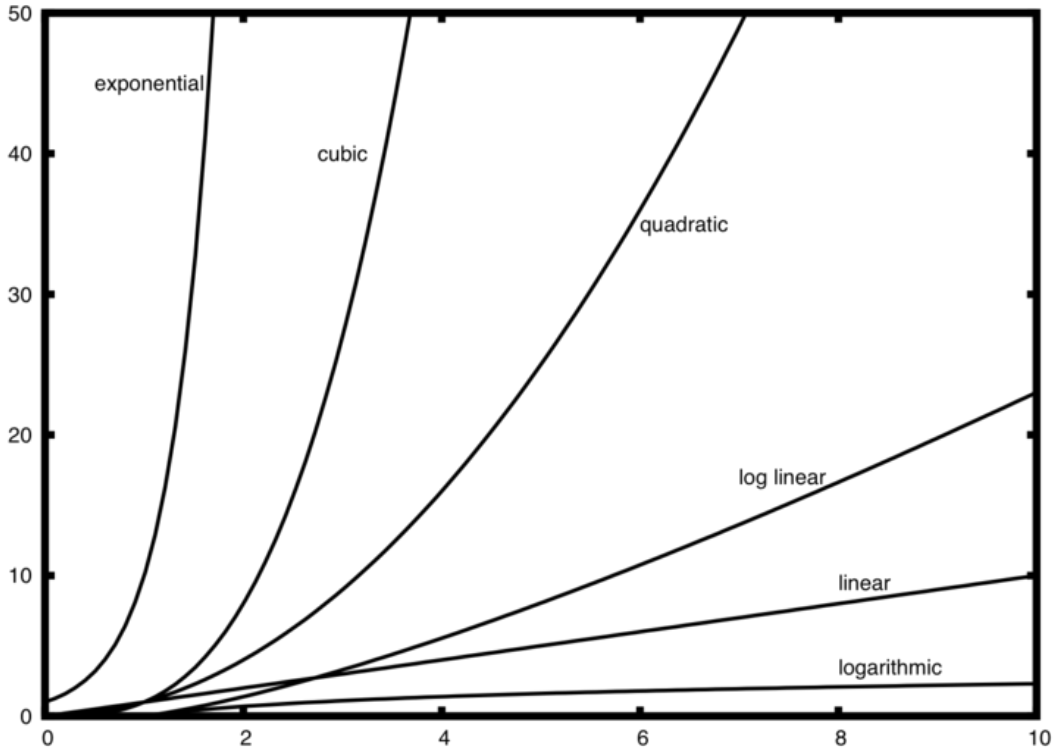
- Testing code
  - Use unit tests
  - Check output of functions with a variety of input values
  - Automate the testing process as much as possible

- Defensive programming
  - Anticipate incorrect data and handle the problem without causing runtime errors

- Runtime errors
  - Exception handling system

Algorithm Analysis

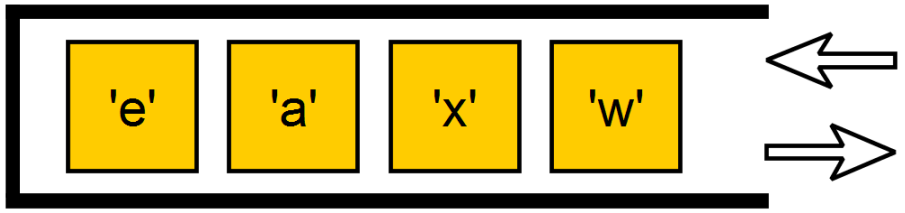
- Used to compare different ways of working with data
  - Interested in scalability – how does time increase with data increase?
  - Write a formula that describes how much time it takes to execute a program in terms of the number of data elements



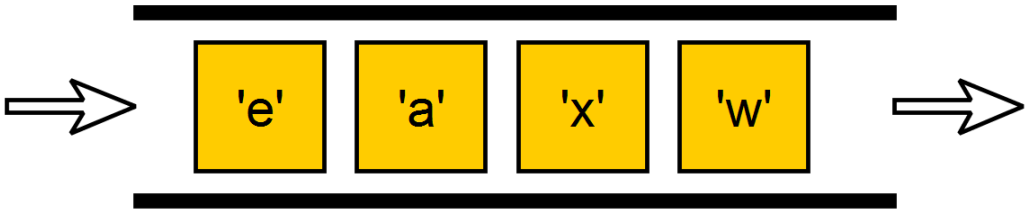
Stacks and queues

- List with constrained access to data
  - Stack adds and removes from the same end of the list
  - Queue adds to one end and removes from the other end

- Stack
  - Last In, First Out



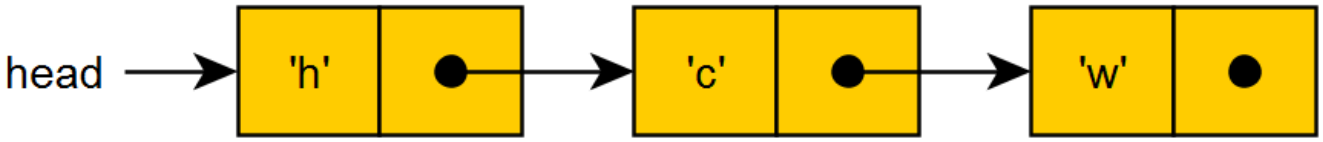
- Queue
  - First in, First out



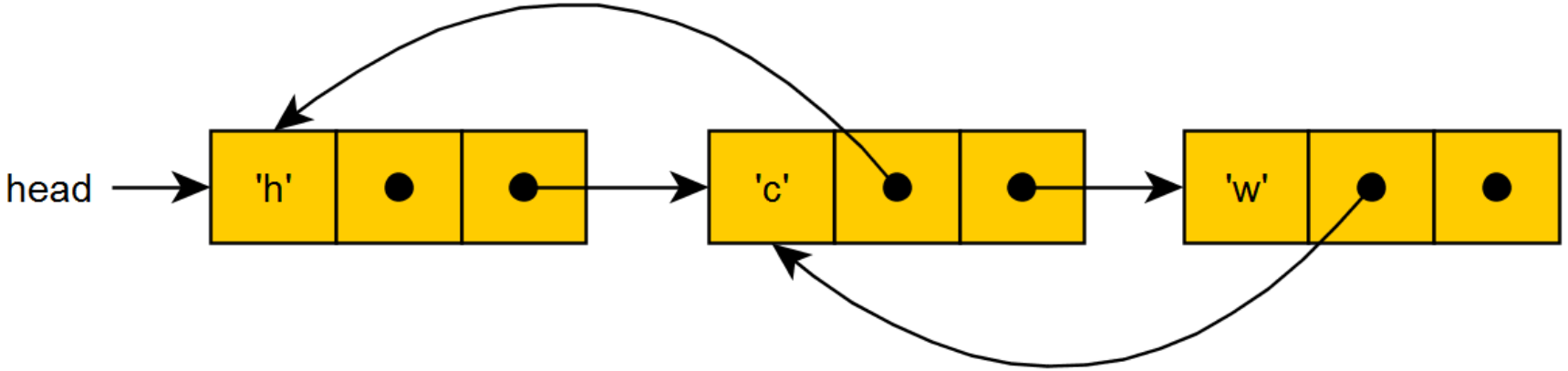
Lists

- Array-based lists

- Singly linked lists



- Doubly linked lists



Recursion

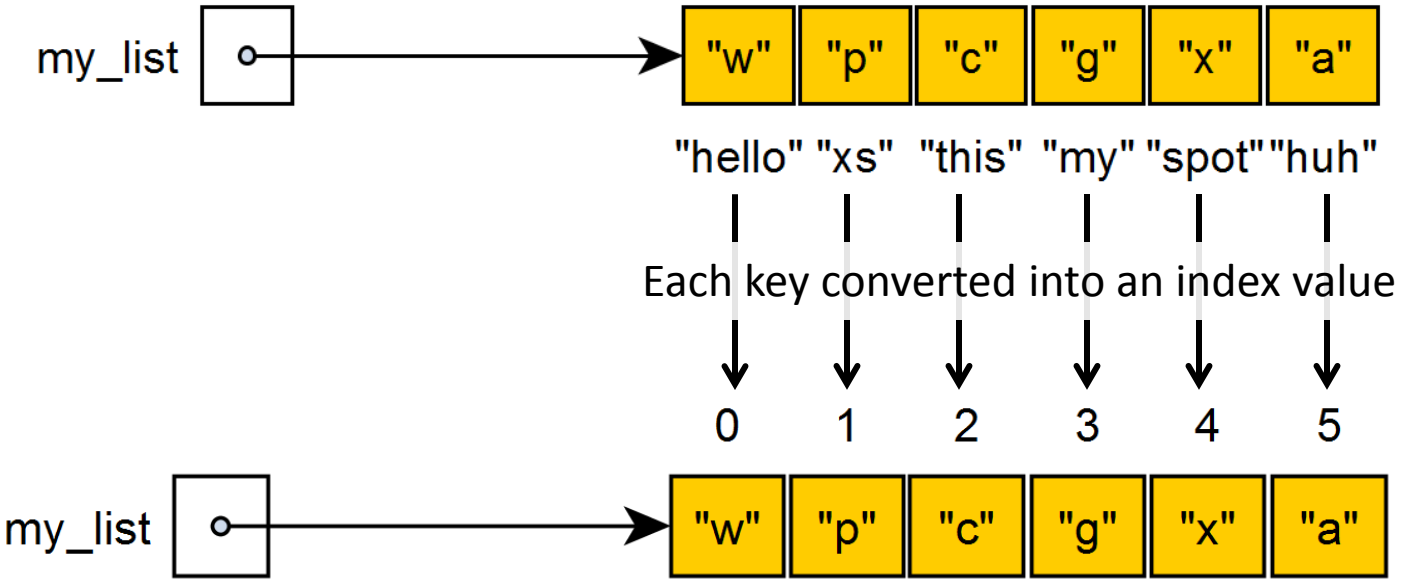
- Programming technique allowing problem solving

$$F(n) = \begin{cases} n & \text{if } n = 0 \text{ or } n = 1; \\ F(n-1) + F(n-2) & \text{if } n \geq 2. \end{cases}$$

```
def fib(n):
    if n == 0 or n == 1:
        return n
    if n >= 2:
        return fib(n - 1) + fib(n - 2)
```

Hash tables

- Storing (key, value) pairs efficiently
  - Take any key and convert into a number between 0 and N - 1
  - Use an array of size N to store the value
  - Deal with conflicts in a consistent way
  - Very fast storage and access
  - BUT – not appropriate for ordered data



Searching

- Linear search
  - Look at each element
  - Works with any kind of data
- Binary search
  - Halve the search space each time
  - Works with ordered data

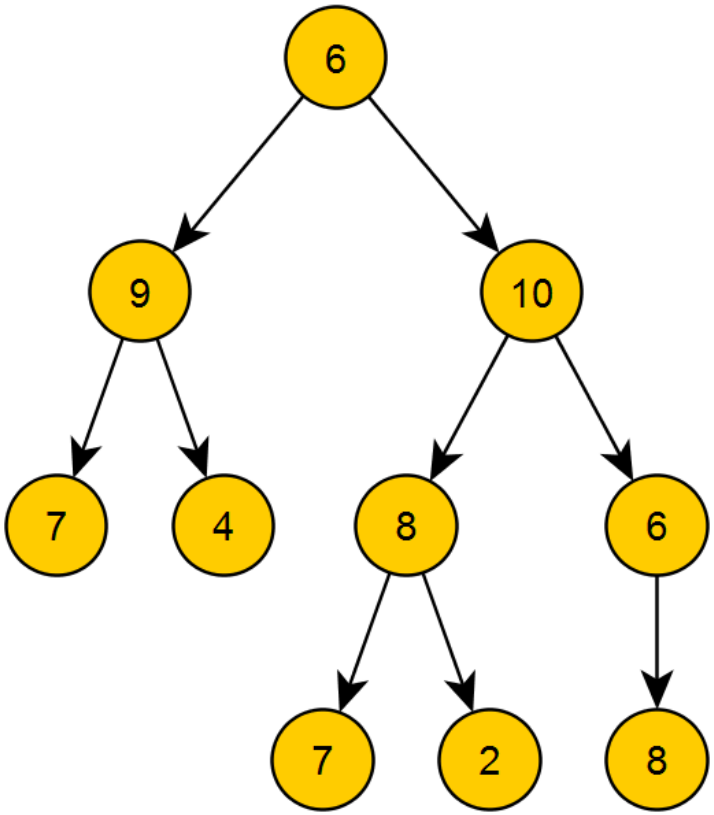
Sorting

- Different sorting algorithms have different trade-offs
  - Overall efficiency
  - Number of comparisons
  - Number of swaps
  - Nature of the data (sorted, unsorted)
  - Stability

http://www.sorting-algorithms.com/

Trees

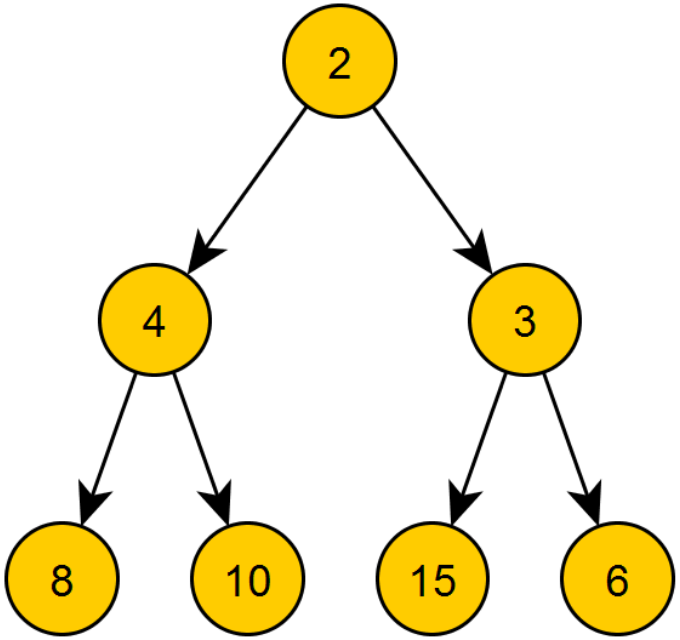
- Recursive data storage
  - Each node has links to other nodes



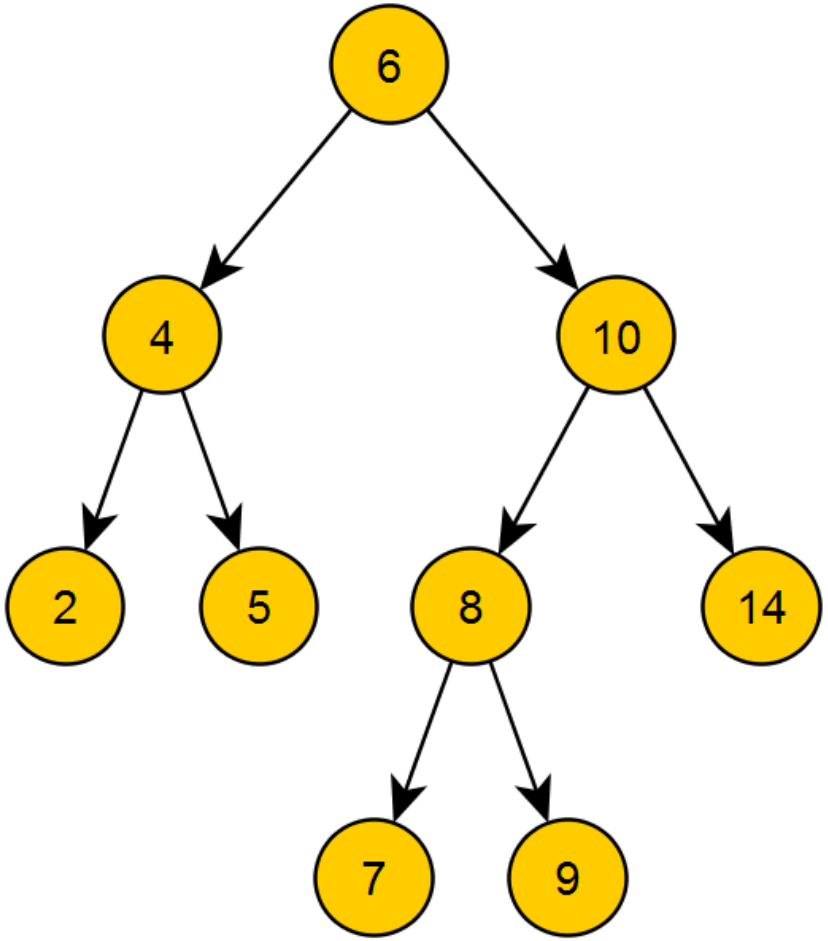
Trees with useful properties

- Storing ordered data

Heap

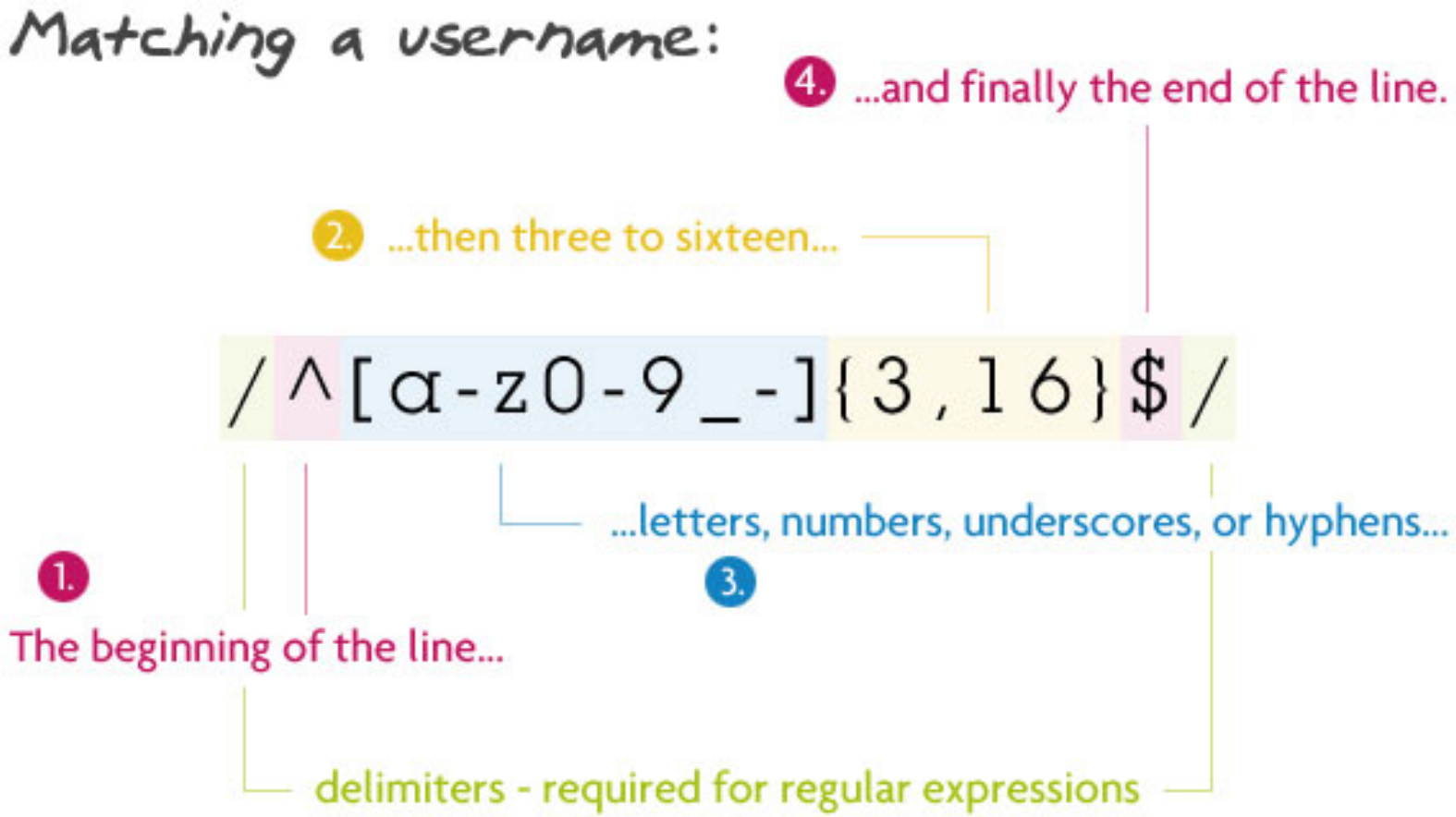


Binary Search Tree



Regular expressions

- Pattern matching text data



Summary

- Introduction to structured programming using Python
- Focus on ways of storing and manipulating data
  - data structures
  - algorithms
- Understanding tradeoffs