CS 1301 Final Exam Review Guide
(Python 3 - Calico)

A. Programming and Algorithm

1. Binary, Octal-decimal, Decimal, and Hexadecimal conversion

- Definition

  Binary (base 2): 10011 = $1*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 1*2^0$
  Octal-decimal (base 8): 725 = $7*8^2 + 2*8^1 + 5*8^0$
  Decimal (base 10): 342 = $3*10^2 + 4*10^1 + 2*10^0$
  Hexadecimal (base 16): C1A = $12*16^2 + 1*16^1 + 10*16^0$

- Conversion

  General process:

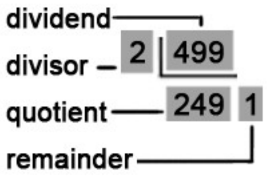
  Decimal ←→ Binary ←→ Octal-decimal / Hexadecimal

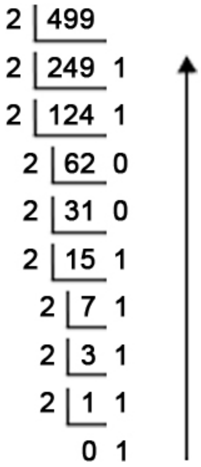
  * You can do octal-decimal / hexadecimal to decimal conversion either by definition directly or by general process (convert to binary version first and than to decimal).

  Decimal → Binary

  Example: 499_{10} → $(\ )_2$

  Step 1: Divide 499 by 2. Write down the result and the remainder as following:





  Step 2: Keep doing this for all the quotients until you get 0 for quotient.
  Step 3: Copy the remainders in a reverse order.
  Result: 499_{10} → 111110011_2

  Binary → Decimal

  Example: 1011101_2 → $(\ )_{10}$
  Step1: calculate by definition

$1011101_2 = 1*2^6 + 0*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0$

Result: $1011101_2 \rightarrow 93_{10}$

Binary → Octal-decimal

Example: $1011101_2 \rightarrow (\ )_8$

Step 1: Divide the binary digits into groups of three from the right side since $8 = 2^3$

$1011101_2 \rightarrow (1)\ (011)\ (101)_2$

Step 2: Convert each group of numbers from binary to decimal by definition

$(1)_2 = 1*2^0 = 1$

$(011)_2 = 0*2^2 + 1*2^1 + 1*2^0 = 3$

$(101)_2 = 1*2^2 + 0*2^1 + 1*2^0 = 5$

Step 3: Combine the conversion results

Result: $1011101_2 \rightarrow 135_8$

Octal-decimal → Binary

Example: $71_8 \rightarrow (\ )_2$

Step 1: Convert each digit to binary (please see the previous 'decimal to binary' part for more details)

7 → 111

1 → 1

Step 2: If some of the results have less than three digits, prepend leading zeroes.

7 → 111

1 → 001

Step 3: Combine the results.

Result: $71_8 \rightarrow 111001_2$

Binary → Hexadecimal

Example: $1011101_2 \rightarrow (\ )_{16}$

Step 1: Divide the binary digits into groups of four from the right side since $16 = 2^4$

$1011101_2 \rightarrow (101)\ (1101)_2$

Step 2: Convert each group of numbers from binary to decimal by definition

$(101)_2 = 1*2^2 + 0*2^1 + 1*2^0 = 5$

$(1101)_2 = 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 13$

Step 3: If some of results are greater than 9, convert them to letters by the following rules:

10 → A, 11 → B, 12 → C, 13 → D, 14 → E, 15 → F

Step 4: Combine the conversion results

Result: $1011101_2 \rightarrow 5D_{16}$

Hexadecimal → Binary

Example: $C2_{16}$ → $(\ )_2$

Step 1: Convert each digit to binary (please see the previous 'decimal to binary' part for more details)

C → 12 →1100

2 → 10

Step 2: If some of the results have less than four digits, prepend leading zeroes.

C → 1100

2 → 0010

Step 3: Combine the results.

Result: $C2_{16}$ → 11000010_2

2. Logical Operation

- **and**

	True	False
True	True	False
False	False	False

- **or**

	True	False
True	True	True
False	True	False

- **not**

not True = False
not False = True

- Order and Priority

Order: left to right
Priority: parentheses > not > and > or

3. Error and try-except

- Three kinds of Error

Syntax Error: An error in a program that makes it impossible to parse — and therefore impossible to interpret.

Runtime Error: An error that does not occur until the program has started to execute but that prevents the program from continuing.

Semantic Error: An error in a program that makes it do something other than what the programmer intended.

- **try-except**

  try:

      block1

  except:

      block2

  block1 will be executed first. If an error occurs during block1's execution, the flow of execution will immediately jump to block2 (skipping any remaining statements in block1). If no error occurs, block2 will be skipped.

4. Conditional Statement

- Conditional Statement: A statement that controls the flow of execution depending on some condition.

- **if**

  if *aBooleanExpression****:***

      statements

  The statements will be executed when the boolean expression is True.

- **elif**

  elif *aBooleanExpression****:***

      statements

  The statements will be executed only when the boolean expression is True and all other previous boolean expressions in the same if group are False.

- **else**

  else:

      statements

  The statements will be executed when all the previous boolean expressions in the same if group are False.

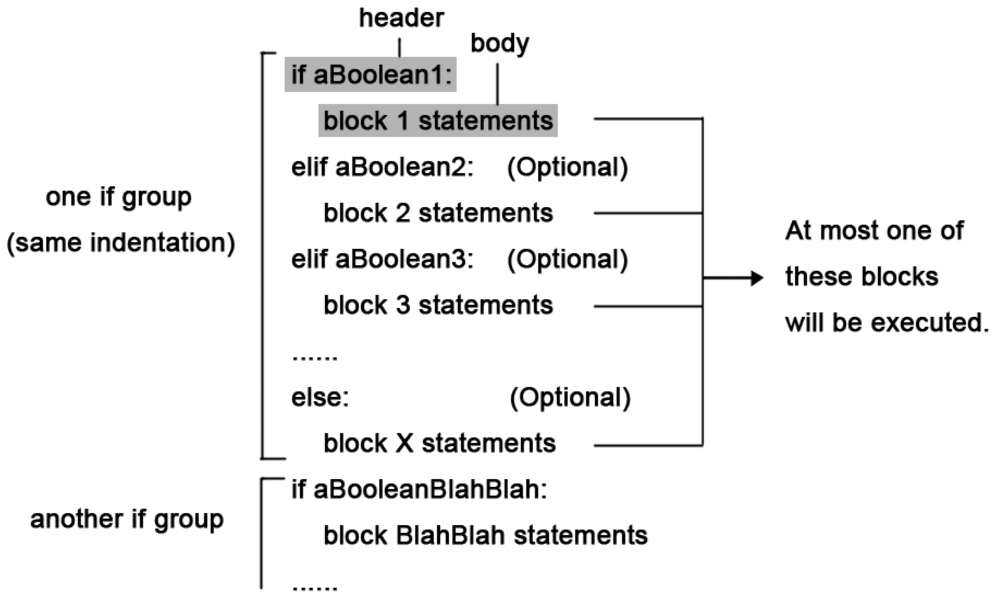
- if Group

  An if group always begins with an if statement, which is the only if statement in the group.

  It might have some elif statements, and at most one else statement at the end.

  All the conditional statements in the same if group must have same indentation.

At most one block of statements will be executed in an if group.



Note: you do not need to have an **elif or **else** statement accompanying every **if** statement.

5. Iteration

- Iteration: Repeated execution of a set of programming statements.

- **for** Loop

  for *item* ***in*** *aSequence****:***

  statements

  The statements will be executed for every item in aSequence. For each execution, the variable identifier defined by the programmer (item) will be assigned to point to the next item in the sequence, and may be used to refer to it in the block of statements. At the end of the for loop, the variable will remain pointing at the last item in the sequence.

- **while** Loop

  While *aBooleanExpression****:***

  statements

  The statements will be executed repeatedly until the Boolean expression becomes False.

  * When using while loop, do not forget to initialize the counter

before the loop and do increment/decrement in the loop

6. Recursion

- Recursion: The process of calling the function that is currently executing.

- Three Elements:

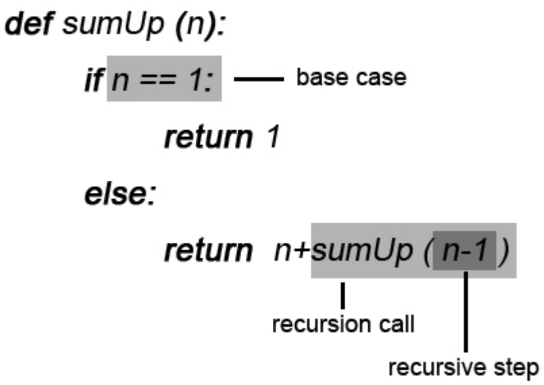
  Base Case: Also know as terminating condition, is a conditional statement that stops recursion at some point and avoids infinite execution.

  Recursion Call: Call the function itself inside the function.

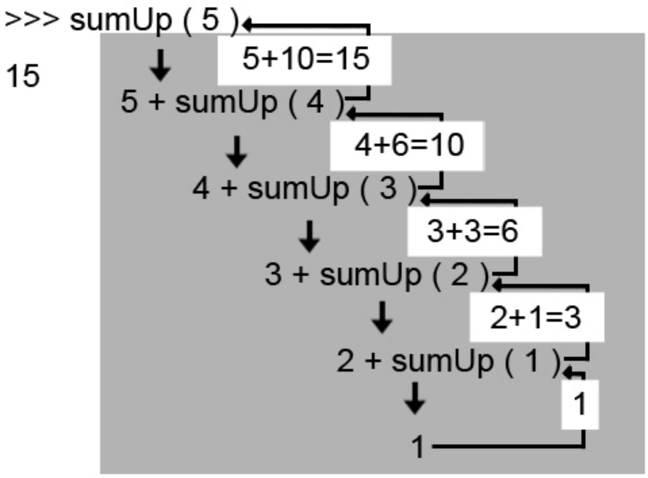
  Recursive Step: The process of approaching the base case. Usually increment or decrement.

- Recursion usually works as iteration. Do not use for loop or while loop together with recursion unless you understand exactly what you are doing.

- Example1:

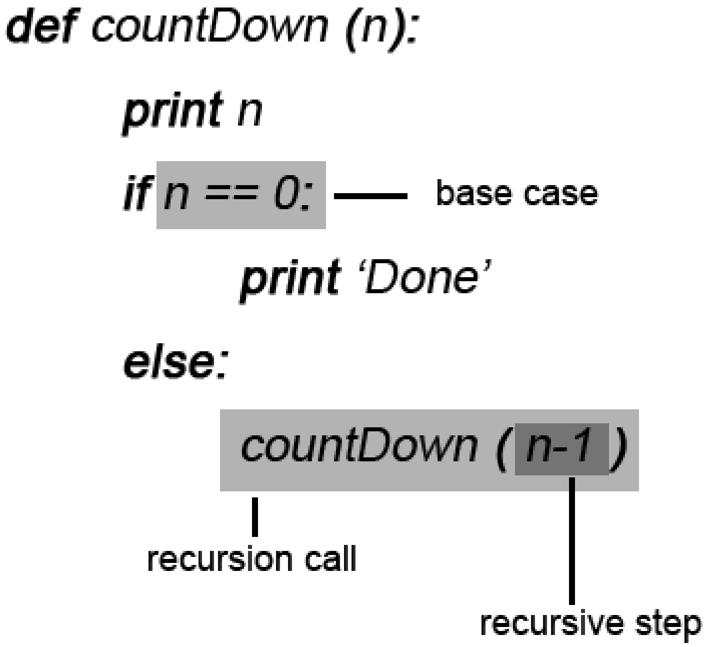


```
def sumUp (n):
    if n == 1:
        return 1
    else:
        return  n+sumUp ( n-1 )
```



```
>>> sumUp ( 5 )
15
```

- Example 2:

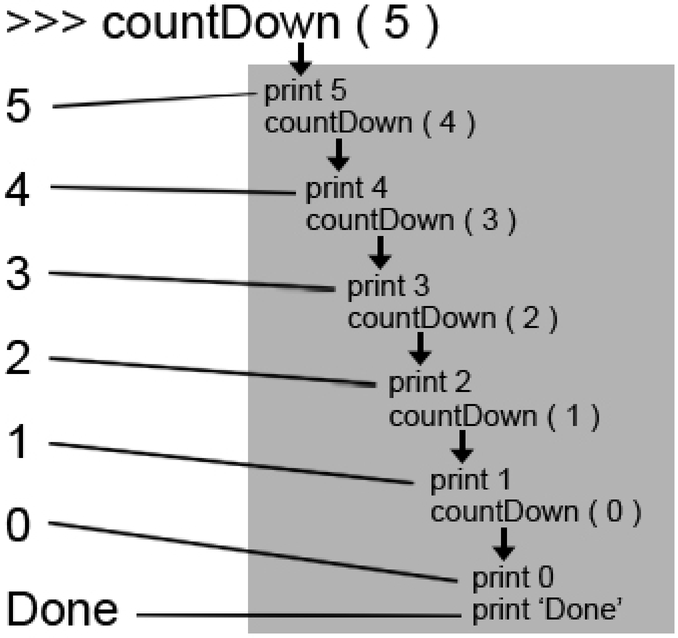


```
def countDown (n):
        print n
        if n == 0:
                print 'Done'
        else:
                countDown ( n-1 )
```

base case

recursion call

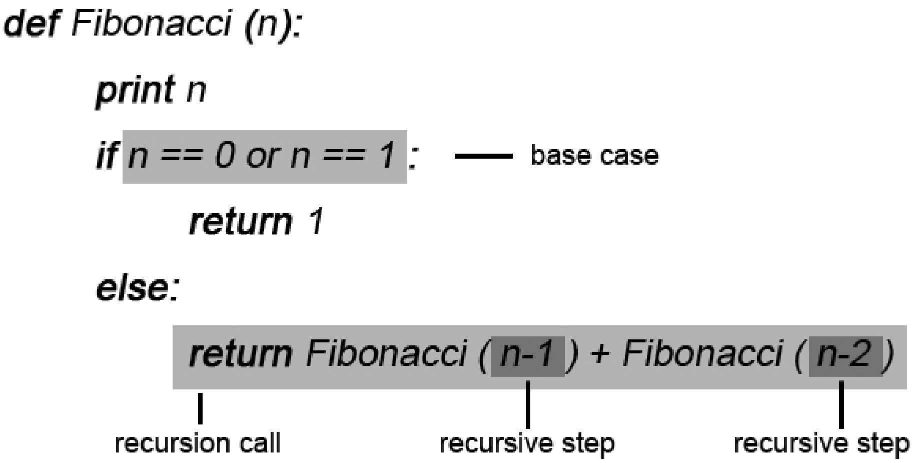
recursive step



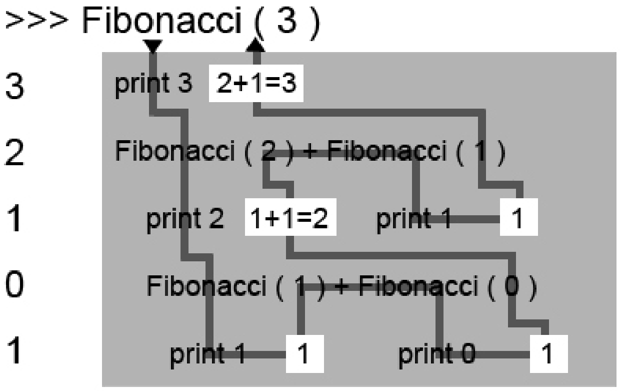
```
>>> countDown ( 5 )
5
4
3
2
1
0
Done
```

print 5
countDown (4)
print 4
countDown (3)
print 3
countDown (2)
print 2
countDown (1)
print 1
countDown (0)
print 0
print 'Done'

- Example 3:



```
def Fibonacci (n):
    print n
    if n == 0 or n == 1 :
        return 1
    else:
        return Fibonacci ( n-1 ) + Fibonacci ( n-2 )
```



```
>>> Fibonacci ( 3 )
```

3
2
1
0
1
print 3
2+1=3
Fibonacci (2) + Fibonacci (1)
print 2
1+1=2
print 1
1
Fibonacci (1) + Fibonacci (0)
print 1
1
print 0
1

7. Functional Programming and lambda Expression

- Functional Programming

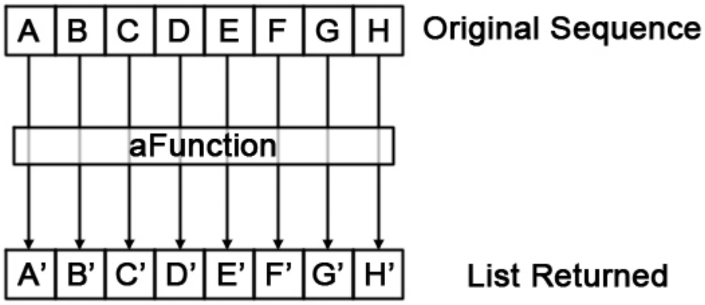
map

map (** aFunction, aSequence **)

map applies the function to all the elements in the sequence and returns a list that has the same length with the original sequence.

aFunction must take in one element.

map returns a list that has the same length as the original sequence, but the elements are modified.



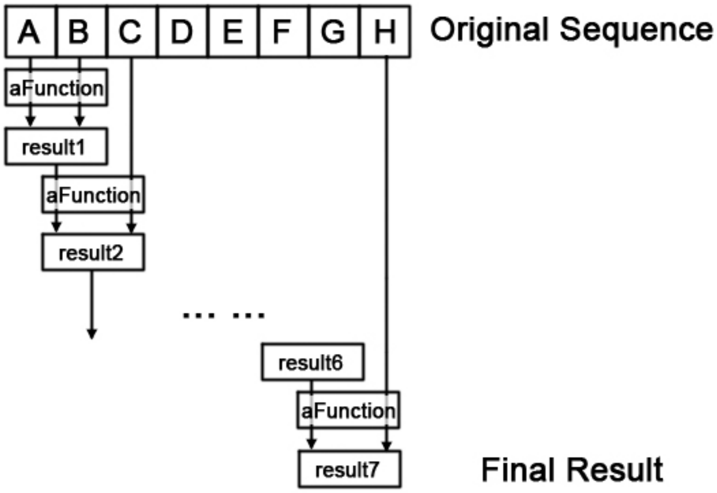
reduce

reduce (** aFunction, aSequence **)

reduce applies the function to the first two elements in the sequence first, and then repeatedly takes in the result that the function returns and the following element as parameters to reduce the length of the sequence, and finally returns one result.

aFunction must take in two elements and return one element.

reduce returns only one element.



filter

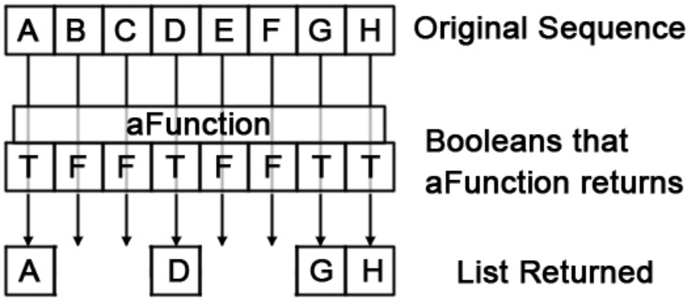
filter (** aFunction, aSequence **)

filter applies the function to every elements in the sequence and gets a boolean. It keeps the element if the boolean is True and removes the element if the Boolean is False.

aFunction must take in one element and return a boolean.

filter returns a new list that is shorter or has the same length as the original sequence, but each element is not

modified.
* filter may return something other than a list. For example, if you filter a string it will return a string.



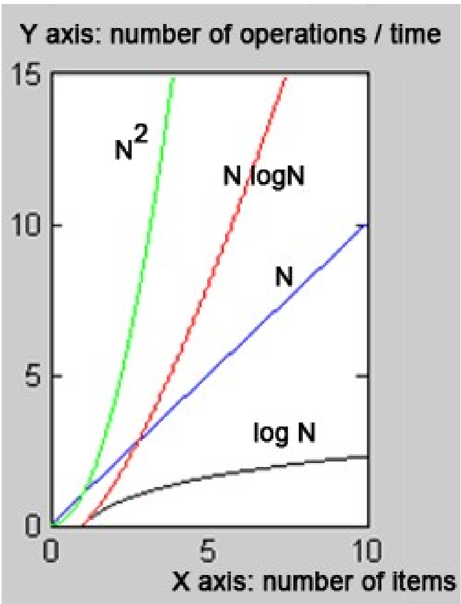
- lambda

  lambda: A block of code which can be executed as if it were a function but without a name.
  lambda *aVariable**:** returnedValue*

8. Search and Sort

- **BigO Complexity**

  BigO notation is used to describe how the work an algorithm does grows as the size of the input grows. In general, you ignore constants when calculating the BigO time complexity of an algorithm.



- **Search**

    a) Linear Search

    Search one by one

    bigO: N (Examine each of the N elements in the list)

    b) Binary Search

    Compare the target value to the mid point of the list. If the mid point is not the target, divide the list in half and try again, searching only the correct half. Repeat until either there are no more elements to check, or until the target is found in the list

- This algorithm can only be performed on sorted lists.

    Example: Search 2 in list [1, 2, 3, 5, 8, 10, 15, 25]

    Round 1:
    Mid point: 8
    2<8
    New list: [1, 2, 3, 5]

    Round 2:
    Mid point: 3
    2<3
    New list: [1, 2]

    Round 3:
    Mid point: 2
    2=2
    Done

    bigO: logN ($\log_2 N$ rounds. 1 comparison each round.)

- **Sort**

    a) Selection Sort (Not required for this course)

    Select the smallest number (if sort increasingly) in the list and append it to the result list.

    Example: Sort [3, 1, 4, 2] increasingly

    Round 1:
    Minimum: 1
    Result list: [1]
    New list: [3, 4, 2]

    Round 2:
    Minimum: 2
    Result list: [1, 2]
    New list: [3, 4]

    Round 3:
    Minimum: 3
    Result list: [1, 2, 3]
    New list: [4]

    Round 4:
    Minimum: 4

Result list: [1, 2, 3, 4]
Done

bigO: N^2 (N rounds. At most N comparisons each round to find out the smallest element.)

b) Insertion Sort

Get the first element in the list. Insert it in the right place in the result list.

Example: Sort [3, 1, 4, 2] increasingly

Round 1:
Element: 3
Result list: [3]
New list: [1, 4, 2]

Round 2:
Element: 1
Result list: [1, 3]
New list: [4, 2]

Round 3:
Element: 4
Result list: [1, 3, 4]
New list: [2]

Round 4:
Element: 2
Result list: [1, 2, 3, 4]
Done

bigO: N^2 (N rounds. At most N comparisons each round to find out the correct location.)

c) Bubble Sort

Pass through the list of elements, and swap adjacent elements if they are not in the correct order. It must repeat the pass N-1 times to guarantee the entire list is sorted (If the smallest element is at the end of the list, it will take N-1 passes to swap it down to the front of the list.)

Example: Sort [3, 1, 4, 2] increasingly

Round 1:
[3, 1, 4, 2] → [1, 3, 4, 2]
[1, 3, 4, 2] → [1, 3, 4, 2]
[1, 3, 4, 2] → [1, 3, 2, **4**]
The last element in the list is guaranteed to be correct after the first round.

Round 2:
[1, 3, 2, **4**] → [1, 3, 2, **4**]
[1, 3, 2, **4**] → [1, 2, **3, 4**]
The last two elements in the list is guaranteed to be correct after the second round.

Round 3:

[1, 2, **3, 4**] → [1, **2, 3, 4**]

The last three elements in the list is guaranteed to be correct after the third round.

Done

bigO: N^2 (N-1 rounds. Each round takes N-1 comparisons. Hence we have (N-1)* (N-1). Because we ignore constants, this is N^2)

d) Merge Sort

Divide the original list into small lists. Merge the small lists.

Example: Sort [3, 1, 4, 2] increasingly

Division stage:

Round 1: [3, 1, 4, 2] → [3, 1] [4, 2]

Round 2: [3, 1] [4, 2] → [3] [1] [4] [2]

Merge stage:

Round 1: [3] [1] [4] [2] → [1, 3] [2, 4]

Round 2: [1, 3] [2, 4] → [1, 2, 3, 4]

bigO: N*logN (log$_2$N rounds and at most N divisions each round in the division stage. log$_2$N rounds and at most N comparisons each round in the division stage.)

e) Quick Sort

Select element as pivot every round and compare the rest elements to the pivot. Elements that are less than the pivot are collected into an unsorted list on the left of the pivot. Elements that are greater than or equal to the pivot are collected into an unsorted list to the right of the pivot. Repeat for the left and right hand collection of numbers until the size of each collection is one, at which point the entire list of numbers is correctly ordered.

Example: Sort [3, 1, 4, 2] increasingly

Round 1:

Pivot: 4 (random choice)

New list: [3, 1, 2, **4**]

Round 2:

Pivot: 1 (random choice)

New list: [**1**, 3, 2, **4**]

Round 3:

Pivot: 2 (random choice)

New list: [**1**, **2**, 3, **4**]

bigO: depends on pivot choices

N*logN (average)

N^2 (maximum)

(Average log$_2$N rounds, at most N rounds. At most N comparisons each round.)

B. Python

1. Data Type

- Basic Data Type

  int (integer)
  float
  bool (boolean)
  NoneType
  *** char** (not really a basic data type, it's really just a special case of string, with length 1.)

- Compound Data Type

  string
  ‘ *aString* **’**
  Elements: characters
  Immutable
  Function:
  *aString.**split (mark)***
  Return a list of the words in the string, using mark as the delimiter string.
  * mark defaults to white space.
  *aString.**find (character)***
  Return the index of character in the string, -1 if not found.
  *aString.**index (character)***
  Return the index of character in the string, VALUE EXCEPTION if not found.

  list
  [*item1, item2,…* **]**
  Elements: any data type (including list, called nested list)
  Mutable
  Functions:
  *aList.**append (item)***
  Add an item at the end of the list.
  *aList.**remove (item)***
  Remove the item in the list.
  *aList.**index (item)***
  Return the index of item in the list, VALUE EXCEPTION if not found.
  *aList.**sort()***
  Sort the list.

  tuple
  (*item1, item2,…* **)**
  * tuple is identified by commas but not parentheses
  Elements: any data type
  Immutable

dict (dictionary)

{ key1: value1, key2: value2, … **}** (Key-Value pairs)

Keys: any immutable data type (including basic data type and immutable compound data type)

* Keys must be different from each other

Values: any data type

Mutable

Function:

*aDict.**get (** key, defaultValue **)***

Return the value of the key in the dictionary.

* defaultValue is optional.

*aDict.**has_key (** aKey **)***

Return a boolean to show whether the key is in the dictionary.

*aDict.**keys()***

Return a list of all the keys in the dictionary.

*aDict.**values()***

Return a list of all the values in the dictionary.

*aDict.**items()***

Return a list of all the key-value pairs in the dictionary. Each item in the list is a tuple in format (key, value).

- Type and Conversion

type (** variable **)

Get the type of a variable

int (** variable **)

Get the integer version of a variable. Variable can be either a float (round down) or a string of integer. (You can't do ***int (** '1.5' **)***).

float (** variable **)

Get the float version of a variable. Variable can be either an integer or a string of number.

bool (** variable **)

Return True or False for a variable. False for 0, None, empty string, empty list, empty tuple, and empty dict., and True for all others. Variable can be any data type.

str (** variable **)

Get the string version of a variable. Variable can be any data type.

list (** variable **)

Get the list version of a variable. Variable can be any compound data type. (get keys for dict.)

tuple (** variable **)

Get the tuple version of a variable. Variable can be any compound data type. (get keys for dict.)

2. Operation

- Order: left to right

- **Priority:** parentheses > indexing > mathematical operation > logical operation > lambda

- Assignment

  aVariable **=** *value*

- Mathematical Operation

  Addition: **+**
  Subtraction: **-**
  Multiplication: *****
  Integer Division: **/** *answer will be an integer
  Floating Point Division: **//** *answer will be a float
  Power: ******
  Modulo: **%**

  * Only when all the numbers used in the operation are integers, the result is an integer, with the exception of the floating-point division. If any of the numbers is float, the result will be a float.
  ex: 11/3 = 3
  11//3 = 3.6
  11/3.0 = 3.6

  Priority: parentheses > power > multiplication, division, modulo > addition, subtraction

- Logical Operation
-

and

	True	False
True	True	False
False	False	False

or

	True	False
True	True	True
False	True	False

not

not True = False
not False = True

Comparisons: **==, !=, <, <=, >, >=**

Sign	**Meaning**
==	Equal to
!=	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

element **in** *aList*:
Will return True or False whether the element is in aList or not.

<u>Priority</u>: parentheses > comparison > in > not > and > or

- Compound Data Type Operation

Concatenation (string, list, and tuple only) :
string1 **+** *string2*
list1 **+** *list2*
tuple1 **+** *tuple2*

ex: “Hello” + “World” → “HelloWorld”

Multiple concatenation (string, list, and tuple only) :
aString ***** *aNum*
aList ***** *aNum*
aTuple ***** *aNum*

ex: “Life”*3 → “LifeLifeLife”

Indexing:
aString ***[*** *index* ***]***
aList ***[*** *index* ***]***
aTuple ***[*** *index* ***]***

aDictionary ***[*** *key* ***]***
* Index starts from 0 in Python.

ex: x = “book” → x[3] = “k”

Remove elements (mutable data types only):

del *aList* ***[*** *index* ***]***
del *aDictionary* ***[*** *key* ***]***
del *aName*

String Formatting

Format operator: The % operator takes a format string and a tuple of values and generates a string by inserting the data values into the format string at the appropriate locations.

“… ***%*** *type1…****%*** *type2…****”*** ***%*** ***(*** *Value1****,*** *Value2****,*** *…****)***

Types:

d and ***i*** for decimal integer
.nf for float with *n* decimal places (***.****n* is optional)
ex: “It is $%.2f” % (14.2345) → “It is $14.23”
s for string

Slicing:

aVariable ***[:]***
Slice everything
Ex: x = “python”
x[:] = “python”

aVariable ***[*** *start* ***:]***
Slice everything after *start*
Ex: x = “python”
x[2:] = “thon”

aVariable ***[:*** *stop* ***]***
Slice everything before *stop*
Ex: x = “python”
x[:4] = “pyth”

aVariable ***[*** *start* ***:*** *stop* ***]***
Slice from *start* (included) to *stop*
Ex: x = “python”
x[1:5] = “ytho”

aVariable ***[*** *start* ***:*** *stop* ***:*** *step* ***]***
Slice from *start* to *stop* with common difference *step*
Ex: x = “python”

x[1:5:2] = “yh”

aVariable ***[: : step]***

Slice everything with common difference *step*

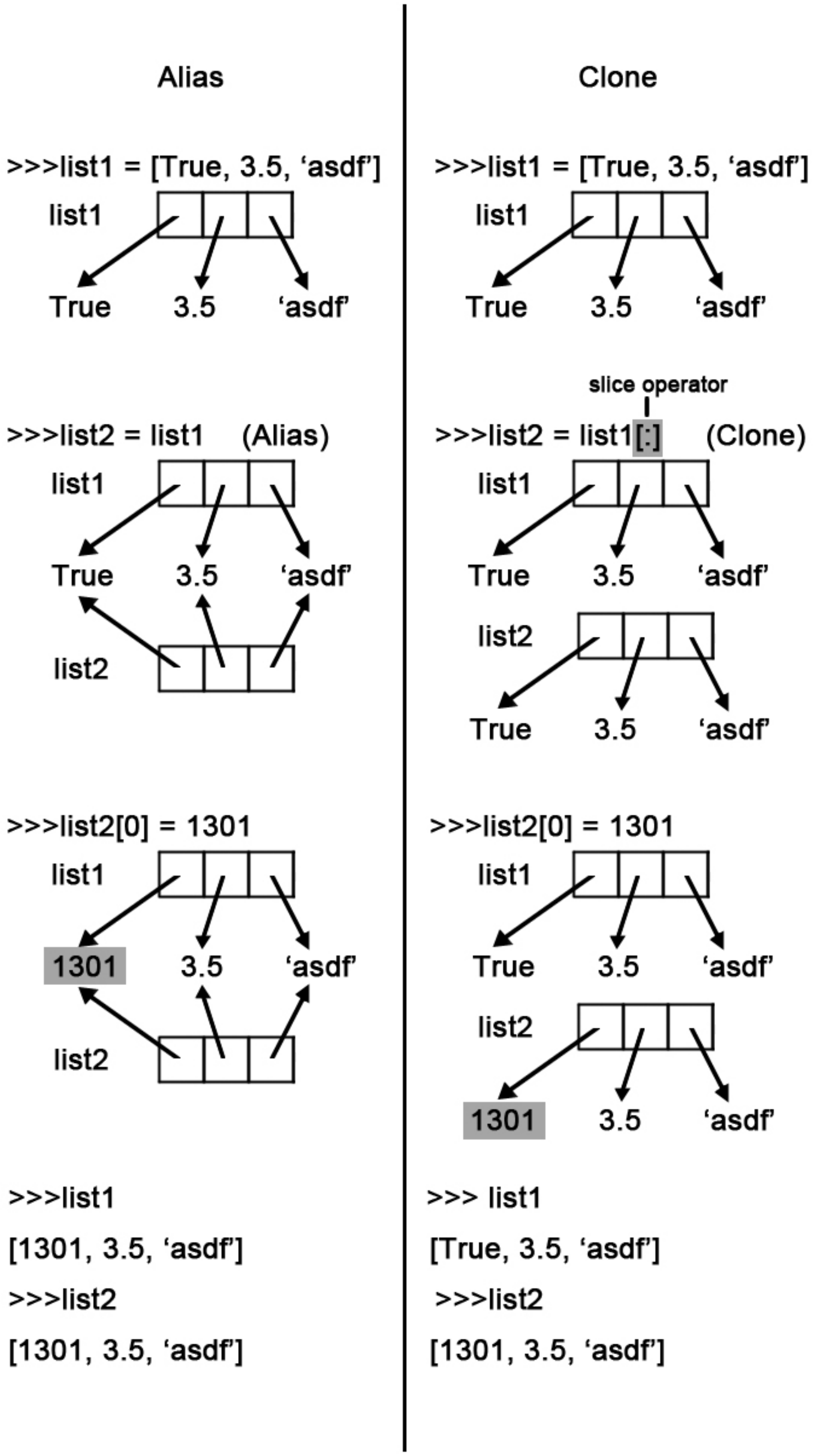
Ex: x = “python”

x[::-1] = “nohtyp”

x[::2] = “pto”

* All the *start*s are included and *stop*s are excluded.

Alias and clone:



Alias

```
>>>list1 = [True, 3.5, ‘asdf’]
```

```
>>>list2 = list1     (Alias)
```

```
>>>list2[0] = 1301
```

```
>>>list1
[1301, 3.5, ‘asdf’]
>>>list2
[1301, 3.5, ‘asdf’]
```

Clone

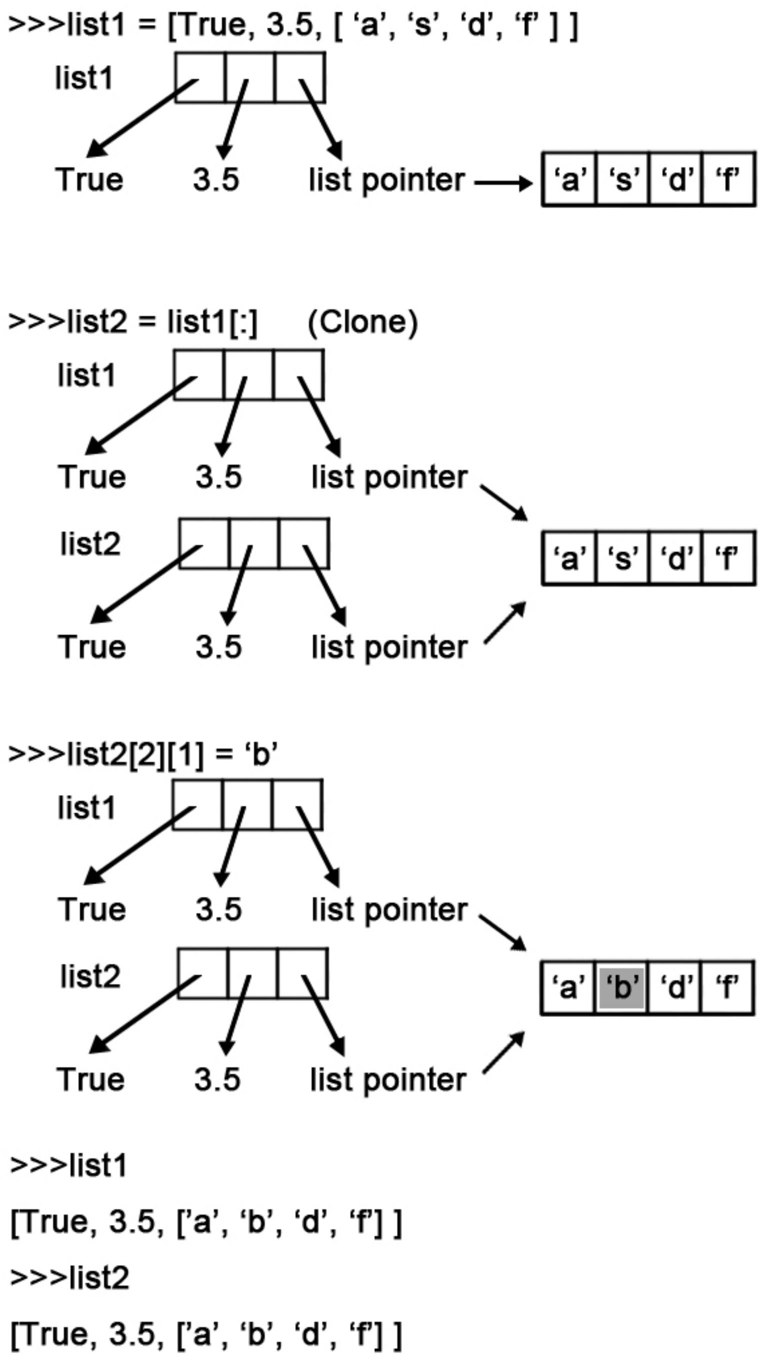
```
>>>list1 = [True, 3.5, ‘asdf’]
```

```
>>>list2 = list1[:]     (Clone)
```

```
>>>list2[0] = 1301
```

```
>>> list1
[True, 3.5, ‘asdf’]
 >>>list2
[1301, 3.5, ‘asdf’]
```

Nested list:



```
>>>list1 = [True, 3.5, [ ‘a’, ‘s’, ‘d’, ‘f’ ] ]
```

```
>>>list2 = list1[:]     (Clone)
```

```
>>>list2[2][1] = ‘b’
```

```
>>>list1
[True, 3.5, [’a’, ‘b’, ‘d’, ‘f’] ]
>>>list2
[True, 3.5, [’a’, ‘b’, ‘d’, ‘f’] ]
```

*Operations on nested lists:

index: Use extra bracket to obtain nested list element

```
ex: aList = [1,2,3, [4,5, [6,7] ,8] ,9]
    aList[3] = [4,5,[6,7],8]
    aList[3][2] = [6,7]
    aList[3][2][0] = 6
```

3. Function

- Define a Function

```
def functionName ( parameter1, parameter2, … ):
    block
    return result
```

- Parameters are optional
- **return** is not required. Default return value in Python is None. Once **return** is executed, the function stops.

* Know the difference between **return** and **print**.

- Print statement:

  In Python 3, print is a function that will “print” something in the command shell.

  Example of correct syntax:

  print(“phone”) → phone

- Global Variable and Local Variable
  - Local variable can be accessed only in the function while global variable can always be accessed.
  - If not declared, all the variables used in function definition are local variables.
  - We can use the key word **global** to make an in-function variable global variable.

    global *aVariable*
  - Example:

```
def foo():
    global aNum
    x = aNum*3
    print(x)
```

*in here, x is a local variable that cannot be accessed outside of the function. aNum, on the other side, is a global variable that is being used inside of the function by calling it with the keyword ***global.***

- Some Useful Functions

  input (*aString* ***)***

  Get something from the user. Always returns a string.

range (** start, stop, step **)

Return a list of numbers that begins at *start* (included) and ends at *stop* (excluded) with common difference *step*
* *start*, *stop*, and *step* here must be integers. start defaults to 0. step defaults to 1.

len (** aCompoundDataType **)

Return the length of a compound data type.

min (** aList **)

Return the minimum value in the list.

max (** aList **)

Return the maximum value in the list.

4. File I/O

- Reading

  myFile= ***open (** filename**, "r")***

  Open the file for reading purpose.

  *myFile.**readline()***

  Return the next line of the file.

  *myFile.**readlines()***

  Return all the lines in the file as a list of strings.

  *myFile.**read()***

  Return all of the contents of the file as a single string.

  *Default mode of file I/O is “r”

- Writing

  myFile = ***open (** filename, **"w")***

  Open the file for writing purpose.

  *myFile**.write (** aString **)***

  Write a string to the file.

  *If a file already exists with the same filename, the old file will be erased and substituted with the newly opened one.

- Appending

  myFile = ***open (** filename, **“a”)***

  Open the file for appending purpose.

- Closing

  *myFile.**close()***

  Close the file. You must do this every time!

C. Robot

0. Import Myro Package and Initialize the robot

- ***from** Myro **import** **

  Import the Myro package

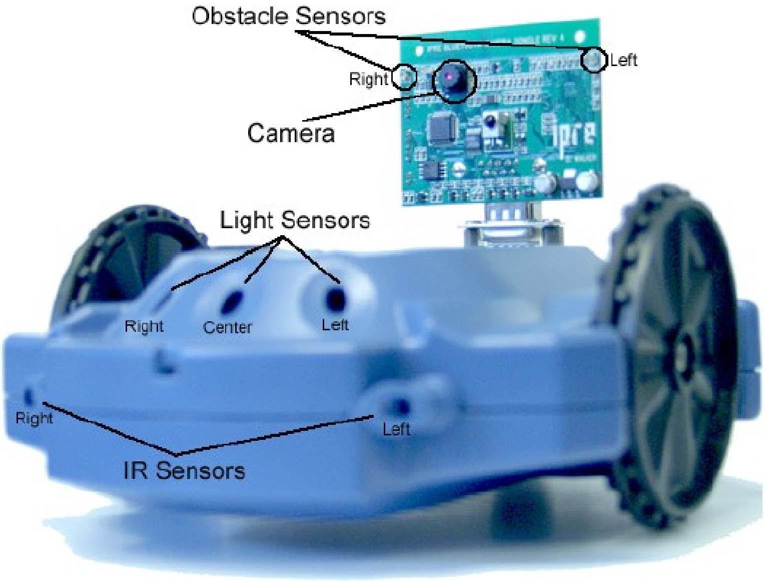
- ***init()***

  Initialize the robot.

- ***setName (** name **)***

  Set the name of the robot.

1. Robot Sensor



- ***getName()***

  Return the name of the robot.

- ***getBattery()***

  Return the voltage of the battery

- ***getLight (** pos **)***

  Read the light sensor. Return an integer which ranges from 0 to 5000, the higher the darker.

  pos can be “left”(0), “middle” / ”center”(1), ”right”(2), defaults to all (returns a list).

- ***getIR (** pos **)***

  Read the IR sensor(s). Return either 0 (there is something in close proximity) or 1 (there is nothing in close proximity).
  pos can be “left”(0), ”right”(1), defaults to all (returns a list).
  *In the back of the robot.

- ***getObstacle (** pos **)***

  Read the obstacle sensor(s). Return an integer which ranges from 0 to 7000, the higher the closer.
  pos can be “left”(0), “middle” / ”center”(1), ”right”(2), defaults to all (returns a list).
  *In the fluke (front of the robot)

- ***takePicture()***

  This function asks the camera takes a picture. Return a picture object.
  *In the fluke (front of the robot)

2. Robot Motion

- Beep

  beep (** duration, frequency1, frequency2 **)
  * *frequency2* is optional

- Movement

  stop()
  Stop the movement of the robot.

  forward (** speed, seconds **)
  backward (** speed, seconds **)
  turnLeft (** speed, seconds **)
  turnRight (** speed, seconds **)

  * *speed* ranges from 0.0 (stop) to 1.0 (full speed).
  * *seconds* are optional in these functions, default to infinite. If *seconds* is not entered, ***stop()*** should be used to stop the robot (usually after ***timer*** for loop).

  translate (** amount **)
  rotate (** amount **)
  move (** translate, rotate **)
  motors (** left, right **)

* *amount* ranges from -1.0 to 1.0.
* Must be used with ***stop()***.

*Timer function (Calico):

Use to make the robot move for a certain amount of seconds. Needs to be used in a for loop.

Example of correct syntax:

```
    for aVariable in timer(seconds):
        do something
```

3. Myro Graphics

- Take, Load, Make, Save, Show Picture

takePicture()
Use the robot camara to take a picture.
*Picture size depends on fluke:
fluke1 = 256 *192.

loadPicture (** aFileName **)
Load a picture file.

makePicture (** width, height, color **)
Make a new picture object.

savePicture (** aPicture / aListOfPictures , aFileName **)
Save a picture or a list of pictures (must be saved as .gif file as animation) as a file.

show (** aPicture **)
Show a picture.

*Remember that, in order to open a picture, the picture must be in your designated python folder (directory). Can use ***getcwd()*** and ***chdir()*** from the **os module** to see/change the directory(folder).

- Edit Picture

Pixel and RGB Color Model

Pixel: Smallest addressable element of a picture.

RGB: The RGB color model is an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors.

Myro Pixel Functions

getPixel (** aPicture, x, y **)
Return a pixel with location (x,y) in a picture.

getPixels (** aPicture **)
Returns an iterator (or generator) that allows you to

iterate through all the pixels using a for loop.

getHeight (**aPicture **)
getWidth (**aPicture **)
Return the height / width of the picture.

getX (**pixel **)
getY (**pixel **)
Return the X / Y position of the pixel.

Myro Color Functions

getRed (**pixel **)
getGreen (**pixel **)
getBlue (**pixel **)
getRGB (**pixel **)
Return the red / green / blue value of the pixel. Range from 0 to 255.

setRed (**pixel, value **)
getGreen (**pixel, value **)
getBlue (**pixel, value **)
setRGB (**pixel, **(** rValue, gValue, bValue **))
Set the red / green / blue value of a pixel.

makeColor (**red, green, blue **)
Return a color.

- Myro Graphics Objects (Paint)
  Calico users: http://calicoproject.org/Calico_Graphics
  *Main difference: need to import Graphics module
  from Graphics import *

  See reference:
  http://www.cc.gatech.edu/classes/AY2011/cs1301_fall/presentations/myro_graphics_reference.pdf

D. Other Stuff You Should Know Before Test

1. Your Name, GT User Name, GTID, Instructor, Section, TA
2. CS History
   http://www.cc.gatech.edu/classes/AY2011/cs1301_fall/presentations/short_history_computing.ppt
3. limited Object-Oriented Programming
4. HTML and CSS
5. Excel, PowerPoint

E. Vocabulary

algorithm: A general process for solving a category of problems.

aliases: Multiple variables that contain references to the same object.

block: A group of consecutive statements with the same indentation.

boolean expression: An expression that is either true or false.

clone: To create a new object that has the same value as an existing object. Copying a reference to an object creates an alias but doesn't clone the object.

compound data type: A data type in which the values are made up of components, or elements, that are themselves values.

conditional statement: A statement that controls the flow of execution depending on some condition. In Python the keywords if, elif, and else are used for conditional statements.

decrement: Decrease by 1.

dictionary: A collection of key-value pairs that maps from keys to values. The keys can be any immutable type, and the values can be any type.

element: One of the values in a list (or other sequence). The bracket operator selects elements of a list.

encapsulate: To divide a large complex program into components (like functions) and isolate the components from each other (by using local variables, for example).

evaluate: To simplify an expression by performing the operations in order to yield a single value.

exception: Another name for a runtime error.

file: A named entity, usually stored on a hard drive, floppy disk, or CD-ROM, that contains a stream of characters.

float: A Python data type which stores floating-point numbers. Floating-point numbers are stored internally in two parts: a base and an exponent. When printed in the standard format, they look like decimal numbers. Beware of rounding errors when you use floats, and remember that they are only approximate values.

flow of execution: The order in which statements are executed during a program run.

format operator: The % operator takes a format string and a tuple of values and generates a string by inserting the data values into the format string at the appropriate locations.

function: A named sequence of statements that performs some useful operation. Functions may or may not take parameters and may or may not produce a result.

global variables: Can be seen through a program module, even inside of functions.

high-level language: A programming language like Python that is designed to be easy for humans to read and write.

immutable type: A data type in which the elements cannot be modified. Assignments to elements or slices of immutable types cause a runtime error.

increment: Both as a noun and as a verb, increment means to increase by 1.

int: A Python data type that holds positive and negative whole numbers.

integer division: An operation that divides one integer by another and yields an integer. Integer division yields only the whole number of times that the numerator is divisible by the denominator and discards any remainder.

Iteration: Repeated execution of a set of programming statements.

keyword: A reserved word that is used by the compiler to parse program; you cannot use keywords like if, def, and while as variable names.

lambda: A block of code which can be executed as if it were a function but without a name.

local variable: A variable defined inside a function. A local variable can only be used inside its function.

low-level language: A programming language that is designed to be easy for a computer to execute; also called machine language or assembly language.

modulo: % operator, that calculates the remainder of an integer division.

mutable type: A data type in which the elements can be modified. All mutable types are compound types. Lists and dictionaries are mutable data types; strings and tuples are not.

nested list: A list that is an element of another list.

None: A special Python value returned by functions that have no return statement, or a return statement without an argument. None is the only value of the type, NoneType operator: A special symbol that represents a simple computation like addition, multiplication, or string concatenation.

parameter: A name used inside a function to refer to the value passed as an argument.

pixel: Smallest addressable element of a picture

proprioception: On a robot, internal sensing mechanisms. On a human, a sense of the relative positions of different parts of ones own body. Example on the robot: Battery or stall sensors.

recursion: The process of calling the function that is currently executing.

robot: A mechanism guided by automatic controls.

runtime error: An error that does not occur until the program has started to execute but that prevents the program from continuing.

semantic error: An error in a program that makes it do something other than what the programmer intended.

sequence: Any of the data types that consist of an ordered set of elements, with each element identified by an index.

short circuit evaluation: When a boolean expression is evaluated the evaluation starts at the left hand expression and proceeds to the right, stopping when it is no longer necessary to evaluate any further to determine the final outcome.

slice: A part of a string (substring) specified by a range of indices. More generally, a subsequence of any sequence type in Python can be created using the slice operator (sequence[start:stop])

str: A Python data type that holds a string of characters.

syntax error: An error in a program that makes it impossible to parse — and therefore impossible to interpret.

traverse: To iterate through the elements of a set, performing a similar operation on each.

type conversion: An explicit statement that takes a value of one type and computes a corresponding value of another type.

variable: A name that refers to a value.

F. Old Exams

Spring 2012

Exam1
http://www.cc.gatech.edu/classes/AY2012/cs1301_spring/codesamples/cs1301-exam1-spring2012.pdf

Exam1 with answers
http://www.cc.gatech.edu/classes/AY2012/cs1301_spring/codesamples/cs1301-exam1-spring2012-answers.pdf

Exam2
http://www.cc.gatech.edu/classes/AY2012/cs1301_spring/codesamples/cs1301-exam2-spring2012.pdf

Exam2 with answers
http://www.cc.gatech.edu/classes/AY2012/cs1301_spring/codesamples/cs1301-exam2-spring2012-answers.pdf

Exam3
http://www.cc.gatech.edu/classes/AY2012/cs1301_spring/codesamples/cs1301-exam3-spring2012.pdf

Exam3 with answers
http://www.cc.gatech.edu/classes/AY2012/cs1301_spring/codesamples/cs1301-exam3-spring2012-answers.pdf

Fall 2011

Exam1
http://www.cc.gatech.edu/classes/AY2012/cs1301ab_fall/codesamples/cs1301-exam1-fall2011.pdf

Exam1 with answers
http://www.cc.gatech.edu/classes/AY2012/cs1301ab_fall/codesamples/cs1301-exam1-fall2011-answers.pdf

Exam2
http://www.cc.gatech.edu/classes/AY2012/cs1301ab_fall/codesamples/cs1301-exam2-fall2011.pdf

Exam2 with answers
http://www.cc.gatech.edu/classes/AY2012/cs1301ab_fall/codesamples/cs1301-exam2-fall2011-answers.pdf

Exam3
http://www.cc.gatech.edu/classes/AY2012/cs1301ab_fall/codesamples/cs1301-exam3-fall2011.pdf

Exam3 with answers
http://www.cc.gatech.edu/classes/AY2012/cs1301ab_fall/codesamples/cs1301-exam3-fall2011-answers.pdf

Fall 2010

Exam1
http://www.cc.gatech.edu/classes/AY2012/cs1301_spring/codesamples/cs1301-exam1-fall2010.pdf

Exam1 with answers
http://www.cc.gatech.edu/classes/AY2012/cs1301_spring/codesamples/cs1301-exam1-fall2010-answers.pdf

Exam2
http://www.cc.gatech.edu/classes/AY2012/cs1301_spring/codesamples/cs1301-exam2-fall2010.pdf
Exam2 with answers
http://www.cc.gatech.edu/classes/AY2012/cs1301_spring/codesamples/cs1301-exam2-fall2010-answers.pdf
Exam3
http://www.cc.gatech.edu/classes/AY2012/cs1301_spring/codesamples/cs1301-exam3-fall2010.pdf
Exam3 with answers
http://www.cc.gatech.edu/classes/AY2012/cs1301_spring/codesamples/cs1301-exam3-fall2010-answers.pdf

Fall 2009

Exam1
http://www.cc.gatech.edu/classes/AY2012/cs1301_spring/codesamples/cs1301-fall09-exam1.pdf
Exam1 with answers
http://www.cc.gatech.edu/classes/AY2012/cs1301_spring/codesamples/cs1301-fall09-exam1-answers.pdf
Exam2
http://www.cc.gatech.edu/classes/AY2012/cs1301_spring/codesamples/cs1301-exam2-fall2009.pdf
Exam2 with answers
http://www.cc.gatech.edu/classes/AY2012/cs1301_spring/codesamples/cs1301-exam2-fall2009-answers.pdf
Exam3
http://www.cc.gatech.edu/classes/AY2012/cs1301_spring/codesamples/cs1301-exam3-fall2009.pdf
Exam3 with answers
http://www.cc.gatech.edu/classes/AY2012/cs1301_spring/codesamples/cs1301-exam3-fall2009-answers.pdf

Important! This is just a review guide which can help you prepare the final. You are responsible for understand everything Jay talked about during lectures this semester. There might be something on the test that is not in this review guide.

Created by Qiqin Xie, Fall 2010
Modified by Cristina Chu, Fall 2012