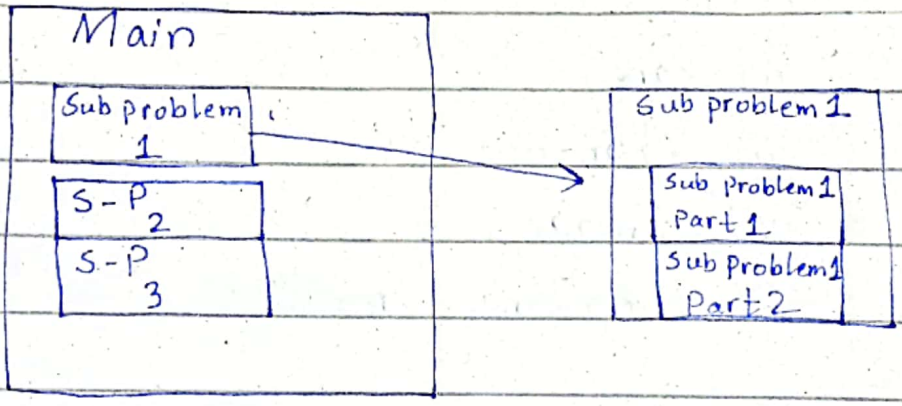
Code in place "CS 106A"

* **Decomposition:-**
  Big problem into smaller pieces.

* **Refinement (step wise):-**
  Problem into pieces and checking with iterative testing where we make sure that smaller pieces of solution are working before move on. Top-down design.



Note: Above mentioned topics are from karel reader.

"Lecture 1:"

Course Plan:

Include
* karel
* Console
* Image data
* Data structure

Assignments:

(i) karel
(ii) Khansole
(iii) Image
(iv) Challenge (creativity)

* karel was used since 80's in Stanford to learn programming.

* karel knows four Commands
  move()
  turnLeft()

Put_beeper()
Pick_beeper()

Defining new commands is the key to write complicated ~~commands~~ programs.

Anatomy of a program:

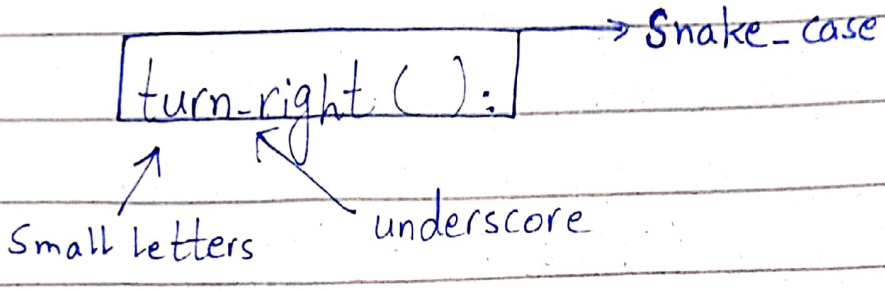
[Import package]
↓
[Main Program] (Function)
↓
[Helper functions]
↓
[Start Program]

"Lecture 2:"

Control flow in karel

* Function name should be descriptive, describe what function do.

* Programs should be readable by humans.



Snake_Case: Basically written in this manner (snake_case). Practice to write code with underscore and no spaces. And with initial letter in lowercased.

For loop:

Repeat for a particular number of times.

```
for i in range(count):
    statement(s)
```
} Syntax

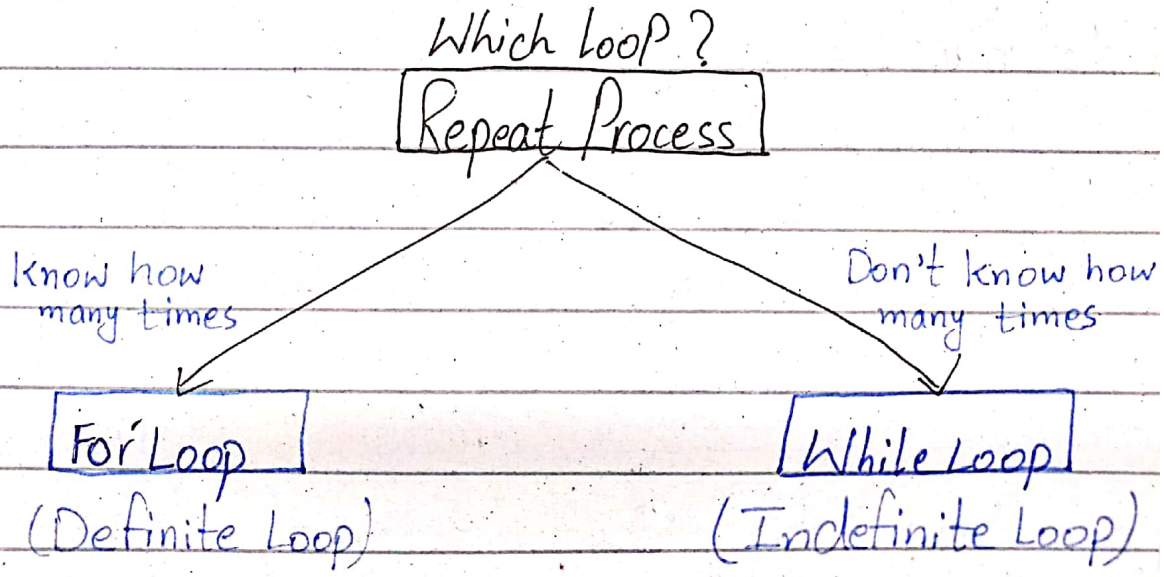
* where i is variable of loop.

While Loop: Works untill condition is true.

```
while condition:
    statements
```

* Buggy program is the program that doesn't work according to the way we intended it to.

* "Off By One Error:"
  OBOE usually comes when last step after while loop required to execute independently. (Similar to inside body of while loop).



if statement:

```
if condition:
    statements
```

if-else statement:

```
if condition:
    statements
else
    statements
```

"Lecture 3":

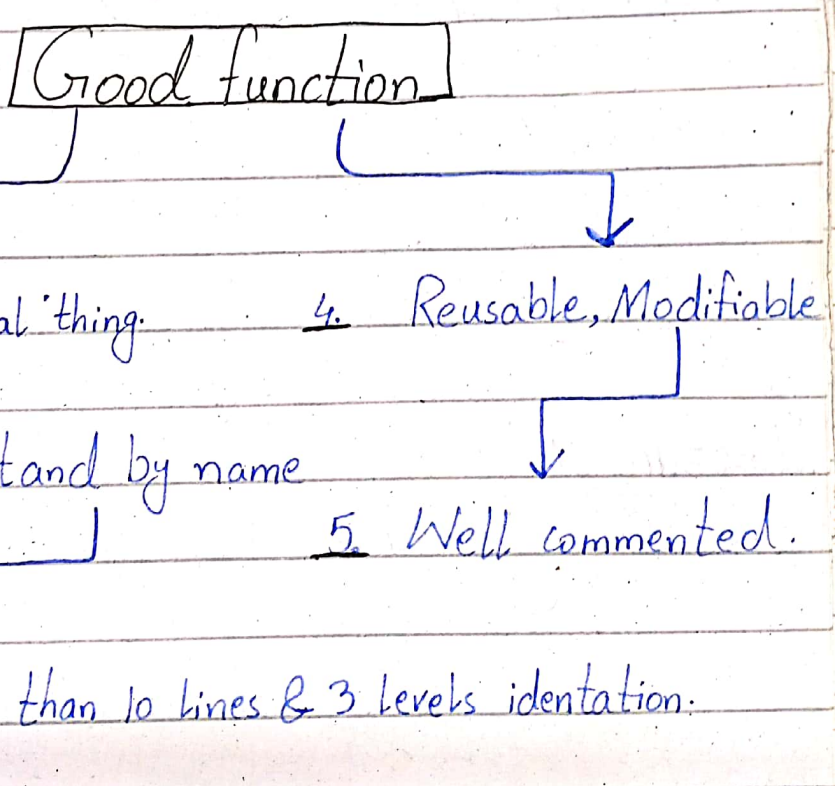
Decomposition

"Top-down approach" or "stepwise refinement"

Break big idea into smaller pieces and then smaller pieces can

also break down further if required.
And then solve one after another.

Decomposition is good to
* solve the problem.
* maintain the problem.
* Understand the solution.



* In programming always does the right thing is more important to use optimization (or do task with fewer steps).

Always work > Fewer steps

Pre condition should be more consistent than number of operations. That's why Algorithm 4 is more consistent than 2.

"Lecture 4:"

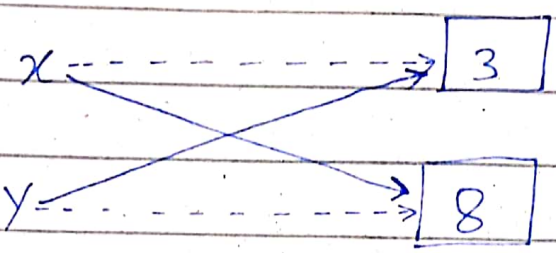
Variables in Python

* function 1-15 lines. (Short functions)

* Python is garbage collecter, if a value is not pointed by a variable Python consider it garbage and

free it from memory. (Python reader)

* If a value is pointed by a variable is swap ~~b~~ with another variable then values remain at same position in memory but pointing of variable changed. (Python reader)



Input from user is in textual form until assign it to a data type.

Scope of variable means, that variable can access or visible inside that particular function.

Variable

(i) contain letter, digits, underscore.
(ii) start with letter or underscore.
(iii) Should be descriptive.

* Variable with data type in print function usually used when there is another string need to print.

"Lecture 5:"

Expressions

(i) 9/2 give float value.

(ii) 9//2 give integer value. (Python reader)

Precedence:

( )	Parenthesis	highest
**	Exponentiation	
-	"Negation" (Unary)	
*, /, //, %		
+, -		lowest

Explicit type conversion:

float(num1) = float

- Num1 is not changed. We create a new value.

int(-2.7) = -2 (int

- Truncate after decimal point. No round off

Constant:

PI = 3.1415
INCHES_IN_FOOT = 12

* Written in all capital SNAKE_CASE.

Math Library:

import math

math.sqrt(x)	Returns square root of x.
math.exp(x)	Returns e^x
math.Log(x)	returns natural log of x.

Random Library:

```
import random
```

	Returns
random.randint(min, max)	b/w min & max min (inclusive) max
random.random()	(float) 0 —— 1
random.uniform(min, max)	Random real number b/w min and max
random.seed(x)	seed of random number generator to x.

"Lecture 6:"

Control flow Python

* while true : ← It remains true by default

* Comparison operator:
  <, >, <=, =>, !=, == (All have equal Precedence)

* If else revisited:
  if (condition)
  elif (condition)
  else:

* Break:
  break statement is used to exit from loop.

* Logical operators:
  not, and, or

* Precedence Madness
  arithmetic > comparison > not > and/or

⋆ Boolean Variables:

```
x= 1<2      # True  (in x)
y= 5.0==4.0 # False (in y)
```

} set after evaluation

```
is_sheltering = True
is_raining = False
```

} Directly set to True or False

```
Play_again = input('Play again? "y" or "n" ')=='y'
if play_again:
```

∴ Note: if play_again = y then it's true otherwise not.

```
is_allowed= not food or drinks
```

not food → False; drinks → True

False or True → True (Because of "or" operator).

As long as you have a drink you should be allowed to bring food in lecture. ☺

* For loop:

```
for i in range(3):
    Print(i * 2)

# equivalently

for i in range(0, 6, 2)
    print(i)
```

start at 0. ↓ (0, 6, 2) ↓ skip by 2 each time.

stop before 6. ↑

result: 0, 2, 4

"Lecture 7:"

Functions

```
def name_of_function(Parameters):
    statements
    #optionally
    return value
```

```
def main()
    mid = average (5.0,10.2)
    print(mid)

def average (a,b):
    sum = a+b
    returne sum/2
```

* It's good practice to store return value in a variable before it's usage.

* Functions can call in print too.
  print(meters_to_cm(5.2))

* When function that is calling, finished it's execution than it releases from memory.

"Lecture 8:"

Functions More Practice
(Functions and parameters)

*
```
def main():
    avg = average(5.0, 10.2)
    print(avg)
```
(5.0, 10.2) — Arguments

```
def average(a, b):
    sum = a+b
    return sum/2
```
(a, b) — Parameters

* On every function call the local variables of that function start fresh mean that no old values disturb them.

* Every time a function is called, new memory is created.

Doctest: >>> represent doctest.

Regression Test

Doctest is a way to test single function at a time, to make sure what they are computing, according to expectation or not?

For doctest, in terminal you can write

py -m doctest fact.py -v

Name of file

Verbose

Works without v but no show result.

* For primitive types (e.g int, float, Boolean, string):
  - copy of values are passed as parameters.
  - Variable passed as an argument is not changed when you modify parameters in function.

Python Reader

<,< = . (Python reader) (1).

Works for strings, list, integer and dates. Uppercase letters are ordered before lowercase.

```
<<< 'apple' < 'banana'
True
```

Python if-de-facto True/False (Python reader)

Python has de facto system, where all empty values count as de-facto false: None, 0, 0.0, empty-list, empty-dict.

```
>>> bool(0.0)
False
>>> bool('hi')
True
```

It helps to test string like following.

```
if len(s)>0:
    Print(s)
```

Boolean (Python reader):

not → and → or

higher to lower precedence (→)

Boolean short circuit:

```
if i<len(s) and s[i].isalpha()...
```

This works. If "i<len(s)" is false the whole expression evaluates to false. In expression "s[i].isalpha()" is not evaluated.

While Loop:

```
while foo()==True:      (Bad)
while foo():            (Good)
```

* Break: Exit from loop

* Continue: Directs the loop run to go back to top of loop immediately to start next iteration.

For Loop:

* range( ): Numbers can be used to execute upto certain extent. (Not including "n" if this is case range(n)).

* We cannot control variable of loop.

```
for num in [2,4,6,8]:
    Print(num)
    num=100   #(No effect on output).
              # Loop reset itself on each
              # iteration.
```

"Lecture 9":

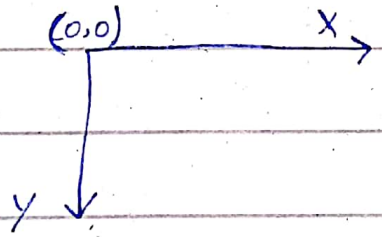
Images

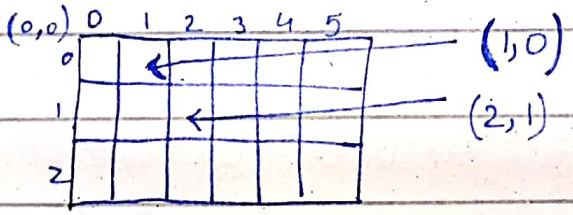
* Global variable is bad style.

Encapsulation principle:

Data used by a function should be a parameter or encapsulated in function.

* Pixel has X and Y coordinates



* 

Pixel (1,0): Red: 6 Green: 250 blue: 7

Due to high intensity of green

, the color is usually of green color.

* We can create new image and set pixels of an image.

For each loop:

```
for item in collection:
    # Do something with item
```

(variable: item; collection of items: collection)

Simple Image properties:

⇒ Image height and width.
⇒ Pixel.x , pixel.y.
⇒ Pixel.red, Pixel.green, pixel.blue etc.

Nested for:

Nested for loop is usefull if we care about x and y.

"Lecture 10:"

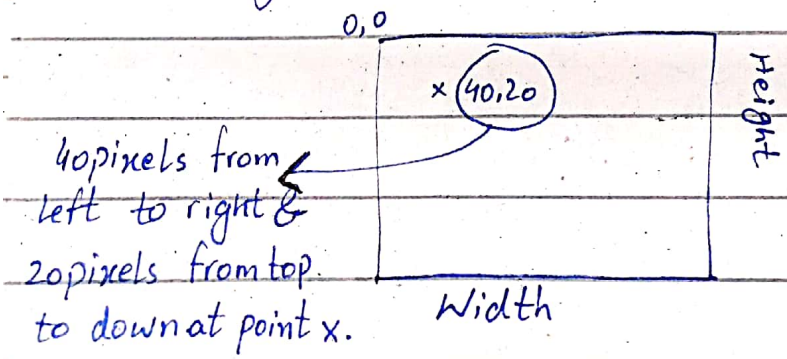
Graphics

* There are mutable & and immutable types of variable in Python
For example "int, string, float, bool" are some of immutable types. In contrast to this there are mutable where we can change the object.

In lecture 10 (Recap or slide67) pixel is pass as an argument to a function named "sepia_pixel" which are modified but don't return, so this pixel object are mutable and change the actual object.

Coordinates:

Coordinates at top left corner is basically a standard.



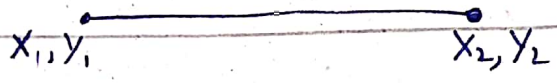
(40, 20)

x, y

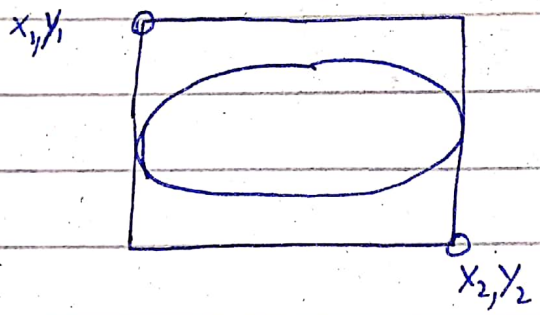
x,y are for width and Height.

Create line:

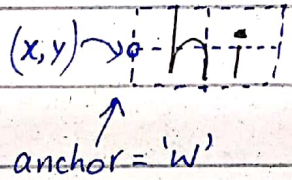
(x1, y1, x2, y2)



Create oval:



Create text:



Tkinter:

Tkinter is a Python graphics Library.

"Lecture 11:"

Animations

(i) Methods are functions bind with a class and can call by object of that class.

(ii) __init__ is a constructor in python. Initialize by using self with dot notation and initilize a variable like

```
self.a = 1
```

or

```
def __init__(self):
    self.x = 1
```

Animations:

Animation is based on frames. It draws a frame, waits for time and draws another frame. Movie and computer game usually draws 50 frames per second.

"Lecture 12:"

Lists

* Python console works in an interactive manner. Console has >>> sign.
exit() is used to leave console.

* None is used to describe "no value" in Python. Comparing anything to None (except None) is False.

None is suitable to use when you would get from function that doesn't return anything.

List:

A list is an ordered collection of items.

↑ refer to elements by there position.

⇒ Dynamically add or remove items.
⇒ Built-in functionality.

* List start/end with brackets. Items separated by commas.

Negative indexes:

```
          -5   -4   -3   -2   -1
letters = ['a', 'b', 'c', 'd', 'e']
           0    1    2    3    4
```

To obtain negative value it works in opposite direction (backward to zero).

* if length of list - Negative value than it gives positive index of same list item.

$5 - 3 = 2 \Rightarrow$ 'c'

letters[-1] is same as letters[len(letters)-1]

letters[-1] ⇓ e

5 - 1 = 4 ⇓ e

List functions:

append (item)	add item to list.
Pop()	Pop value at specific index.
remove (elem)	Remove value in list.
extend (list)	All elements of one list add to another list.
+	Concatenate multiple lists and assign to a new list.

All other lists remain unchanged.

Index (elem)	Return element's index which is at initial location of match.
insert (index, elem)	Insert element at given index and shifts all other elements.
copy( )	To copy list.
max (list)	Returns maximum value of list.
min (list)	Returns minimum value of list.
sum (list)	Returns sum of values of list.

Looping:

For each loop

```
for elem in str_list:
    print(elem)
```

For loop

or

```
for i in range(len(str_list)):
    elem = str_list[i]
    print(elem)
```

List as a parameters: in a function

Creating a new list means you're no longer dealing with list passed in as parameter. At that point you are no longer changing parameter passed in.

```
def main():
    values = [5,6,7,8]
    create_new_list(values)
    print(values)

def create_new_list(num_list):
    num_list.append(9)
    num_list = [1,2,3]
```

Output: [5,6,7,8,9]

Note on Loops and lists:

* For Loop using range:

```
                                list=[1,2,3]
for i in range(len(list)):
    list[i] += 1    # Modify list
```

output: [2,3,4]

Always work for primitive (integer, float) and non-primitive (Pixels) data types.

Often use for loop with range when elements are primitive type.

For-each loop:

```
for elem in list:          # Modify local variable
    elem += 1              # Not changing list.
```

For each loop often use when elements in list don't need to modify mean it's suitable to use for each loop ~~me~~ when working with non-primitive data types.

Otherwise above expression change the element variable not items inside list. Because element variable work as an integer counter 0, 1, 2 .....

"Lecture 13:"

Text Processing

All characters in a string have an index. A string is indexed just like a list. It is like list of characters.

x = 'this is a Test'

Strings function:

string functions given below:

split	x.split(' ')	# split words ~~spaces~~
Upper	x.upper()	# Upper case letters.
Lower	x.lower()	# Lower case letters.
replace	x.replace('is', 'lol')	# replace is with lol
find	x.find('is')	#
strip	strip()	# Remove spaces either side.

Note: Above strings lie in must know category.

Good to Know:

startswith	x.startswith('th')	# True (it starts with th)
endswith	x.endswith('end')	# False
title	x.title()	# 'This Is A Test'
isalpha	x.isalpha()	# False (No character in string)
isdigit		# Check digit in string.
isspace	' '.isspace	# If there's any space. (True)

Computational Biology:

DNA → mRNA → Protein.

Function which takes strings:

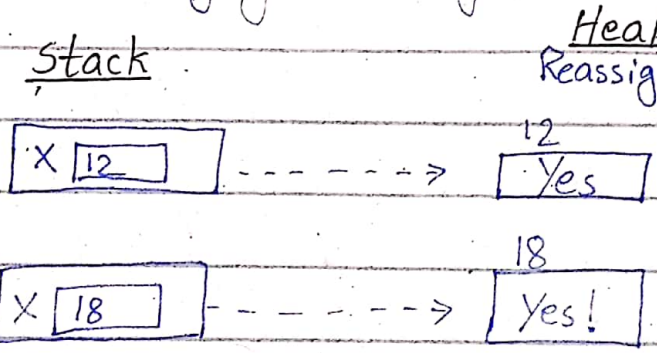
```
ord('A')  # Gives ASCII of character.
hash(x)   # Gives consistent numbers seq. every time.
```

Strings are Immutable:

Strings are immutable in Python.

→ Reassign the string is allowed.
→ Concatenate can be possible.

In above two situations strings are created and then reassign or concatenate by changing the reference in memory instead of modifying the original string.



"Lecture 14:"

Dictionaries

* Dictionaries are based on key and value.

Example: (real world)

key : name

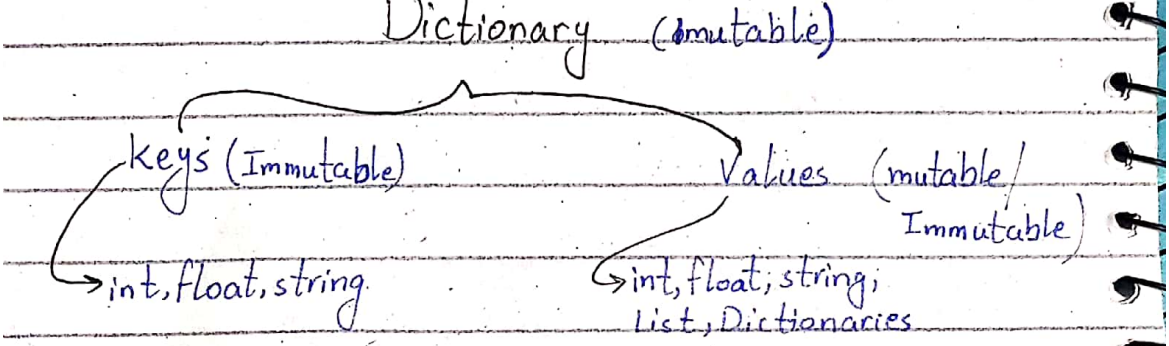
values: Phone number.

* Represent in braces { }
* key and value pairs separated by colon : key and value is considered as single pair. And after each pair comma is used.

Syntax example:

ages = {'chris' : 32, 'Brahm' : 23, 'Mehran' : 50}

keys have to be unique.



Dictionary functions:

- dict.get(key) — Return value of associate key.
- dict.get(key, default) — If key not present default will show.
- dict.key() — Return something similar to range of keys.
- dict.values() — Return something similar to range of values.
- dict.pop(key) — Remove key/value pair of a key. Returns value of that key too.
- dict.clear() — Remove all key/value pairs in dictionary.

len(dict) Return key/value pairs length.

del dict[key] Remove key/value pairs in dictionary and doesn't return value like pop.

Note: In dictionary keys are unique so adding similar another key overwrite the original ones. (just value)