

THE ONLY SHIBBOLETH THE WEST HAS IS SCIENCE. IT IS THE PREMISE OF MODERNITY AND IT DEFINES ITSELF AS A RATIONALITY CAPABLE OF, INDEED REQUIRING SEPARATION FROM POLITICS, RELIGION AND REALLY, SOCIETY. MODERNISATION IS TO WORK TOWARDS THIS.

BRUNO LATOUR

THE BOUNDARY BETWEEN SCIENCE FICTION AND SOCIAL REALITY IS AN OPTICAL ILLUSION.

DONNA HARAWAY

CSEF

INTRODUCTION TO PYTHON

THE STUDENT ACADEMY

Contents

<i>Introduction to OSX Terminal</i>	15
<i>Git</i>	17
<i>Git</i>	21
<i>LaTeX</i>	25
<i>Integers and Floating Point Numbers</i>	29
<i>The Boolean</i>	33
<i>Conditional Statements</i>	37
<i>While Loops</i>	39
<i>While Loops</i>	43
<i>Bases</i>	45
<i>Array 1-D, 2-D, 3-D</i>	49

Arrays 53

For Loops, Nested For Loops 57

Libraries 63

Random 65

Statistics Library 67

Statistics 73

Graphics Library 75

Graphics 79

Monte Carlo 83

Game of Life 85

List of Figures

1	This is the logo of latex.	23
2	this is what latex paper can look like.	24
3	The ultimate answer of the universe is also an Integer!	29
4	Flow diagram about how the while loop works	39
5	The Universe	60
6	More probability density is found as one gets closer to the expected (mean) value in a normal distribution. Statistics used in standardized testing assessment are shown. The scales include standard deviations, cumulative percentages, percentile equivalents, Z-scores, T-scores, standard nines, and percentages in standard nines..	73
7	This is an example of python graphics.	79
8	This is an example of python graphics, Squares.	79
9	This is an example of python graphics,10 circles.	80
10	This is a type of game of life.	88

List of Tables

- | | | |
|---|---|----|
| 1 | A list of Unix shell commands. | 18 |
| 2 | A list of git commands for version control. | 19 |

*The longest snake ever held captive is Medusa,
a reticulated python (python reticulatus).*

*On 12 October 2011, she was measured at
7.67 m long.*

Note

This physics text is an OpenSource academic project developed in abstraction at The Academy. The manuscript is written in \LaTeX and makes use of the `tufte-book` and `tufte-handout` document classes.

<http://latex-project.org/ftp.html>

<https://git-scm.com/downloads>

Introduction to OSX

Terminal

Intro

Terminal (Terminal.app) is the terminal emulator included in the OS X operating system by Apple. (Wikipedia)

A terminal emulator is an application that allows access to the operating system. This emulator can be used to perform multiple tasks; it can perform from simple tasks like printing out text, to performing more complex tasks like uploading files to the cloud.

Basic Commands:

break : Exit from a For, While, Until or Select loop.
cd : Change Directory
chmod : Change access permissions
clear : Clears terminal screen
curl : Transfer data from or to a server
date : Display or change the date & time
echo : Display text on screen
exec : Execute a command
false : Do nothing, unsuccessfully
for : Loop command
info : Help info
jobs : List active jobs
reboot : Stop and restart the system
shutdown : Shutdown or restart OS X
sleep : Delay for a specified time
until : Loop command
while : Loop command
!! : Run the last command again

Hello World

Here is an example HelloWorld Program written for OS X Terminal.

```
echo Hello World
```

```
Hello World
```


Git

Git is a distributed revision control system with an emphasis on speed, data integrity, and support for distributed, non-linear workflows. Git was initially designed and developed by Linus Torvalds for Linux kernel development in 2005, and has since become one of the most widely adopted version control systems for software development.

Please note the below assume you are using a Terminal shell in Linux or OSX operating system. If you are using Windows you will use "dir" instead of "ls" to list files using Command Terminal. Also note the slashes are different for writing file paths. Linux and OSX use forward slash / while Windows uses back slash.



Introduce yourself to Git

```
$ git config --global user.name "Dr Doeg"  
$ git config --global user.email doeg@example.com
```

This is the first step you must do when using git for the first time. Tag your commits with Name and Email.

Basic Terminal Commands

Using the terminal you may navigate the file directory. Make, delete, move and rename files and directories.

```
$ cd path/to/project/folder
$ ls
$ cp filename ~/Location/newname
$ mv filename ~/Location/newname
$ rm filename
$ rmdir directoryname
$ touch filename
$ mkdir directoryname
$ nano filename
```

Unix Command	Action
cd	change directory
ls	list files
cp	list files
mv	move files
rm	remove files
rmdir	remove directory
touch	create file
nano	edit file
mkdir	create directory

Table 1: A list of Unix shell commands.

Git Repository

You can get a Git project using two main approaches. The first takes an existing project or directory and imports it into Git. The second clones an existing Git repository from another server.

Setting up a local Repository

```
$ git init
```

Using the command line navigate to the project folder and initialize a git repository.

```
$ git add file2.jpg
$ git add .
```

Add files in the folder to the stage.

Or add all the files.

```
git commit -m "comment on the file changes"
```

Commit the additions.

Push Your Local Repository to GitHub

```
$ git remote add origin https://github.com/<USER>/<REPO>.git
```

Setup the remote repository location on GitHub using your account.

```
$ git remote set-url origin https://.../<USER>/<REPO>.git
```

If you already set up the remote and want to change it use "set-url".

```
$ git push origin master
```

Push the committed structure to the remote server.

Cloning an Existing Repository From GitHub

```
$ cd path/to/whereUwant/folder
```

Navigate to the desired location in file structure.

```
$ git clone https://github.com/<USER>/<REPO>.git
```

Set the location on the GitHub server to place the repository.

Working With Branches

Version control is one of the great powers of git.

```
$ git branch
$ git branch branchname
$ git checkout branchname
$ git merge branchname
$ git branch -m newbranchname
$ git branch -D branchname
```

Unix Command	Action
branch	list branches
branch <NAME>	create new branch
checkout <NAME>	switch to new branch
merge <NAME>	merge branch with current
branch -m <NAME>	rename current branch
branch -D <NAME>	delete branch

Table 2: A list of git commands for version control.

Updating an Existing Repository From GitHub

```
$ git fetch  
$ git pull -u origin master
```

The sophisticated way to update uses `fetch`, reviews changes and merges those onto the master branch. The `alt` the current project folder from the GitHub remote server.

Get Git, Github and More on Git

<https://git-scm.com/downloads>
<https://github.com/>
<https://git-scm.com/book/en/v2>

Download git and register an account at GitHub. Look at the official documentation for more information.

Git

What the git is

Github is a website that can make people able to share their ideas. People can upload and download by pushing and cloning in order. When the git saves the files, git gets the checksum with sha-1 hash before it saves. Therefore, it is difficult to lose and check the files. This checksum is a 40-character string composed of hexadecimal characters and calculated based on the contents of a file or directory structure in Git.

Getting a git repository

There are two main approaches getting a git repository. The first one take an existing project or directory it into git. The second one clone an existing git repository from another server.

If you want to move an existing project in git, you need to go to the project's directory and type:

```
-git init  
-git add .  
-git commit -m 'comment on the file changes'
```

If you want to clone a git project, you need to do so like this:

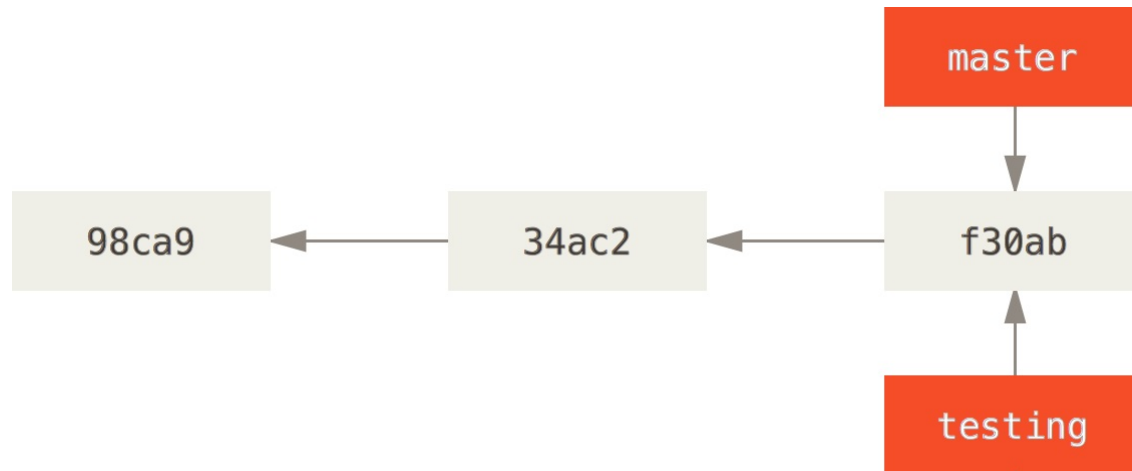
```
-git clone https://github.com/YOUR-USERNAME/YOUR-  
REPOSITORY
```

Branch

The git branch command is a branch administration tool. This command can show the list of all branch, create a new branch, delete a branch and rename a branch

If you want to create a new branch, The git branch command can make a testing branch:

```
- git branch testing
```



Git is different from the other version administration tool. Git has a special point called HEAD. This is a pointer to local branch you're currently on. In this case, your git is still on master branch. The git branch command makes a new branch, doesn't move the branch.

Merge

The git merge command merges other branch into the branch you have checked out. If you use the "git merge <branch>" command, the branch would merge.

what is latex

LaTeX is a document preparation system for high-quality typesetting. It is most often used for medium-to-large technical or scientific documents but it can be used for almost any form of publishing. (Basically its a high-tech version of a pdf document maker that can do much more stuff than normal word documents.)

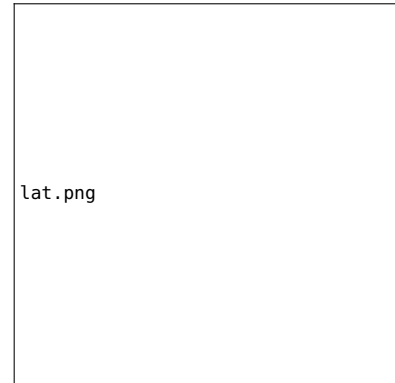


Figure 1: This is the logo of latex.

this is how you start a latex file.

```
\documentclass{tufte-handout}

\title{Latex}

\author[The Academy]{Tony/Zekang Lin}

\begin{document}
```

Here you start you file by telling what kind of file you are making and the title and author of it. And begin the document.

And begin the document. remember to have enddocument at the end of the paper.

this is some useful things you can do in latex file.

```
\begin{marginfigure}%
\includegraphics[width=\linewidth]{XXXXX.png}
\caption{This is the logo of latex.}
\label{fig:marginfig}
\end{marginfigure}

\marginnote[30pt]{DXXXXXXXXXXXXXXXXXXXXXXXXXXXXX}

\begin{shaded}
\begin{verbatim}
```

you can use margin figure or figure to include pictures as a note or just something that you want to show. it will auto matically lable the picture as figure + the number of the image.

use marginnote to add notes.

shaded can help you shade what you are going to write and verbatim will allow you to write your codes in latex.

why do we use latex.

Latex is easy to use and there are many stuff that latex will automatically do for you, such as it will automatically write the date that you last edited and automatically label the number of images you added to the paper.

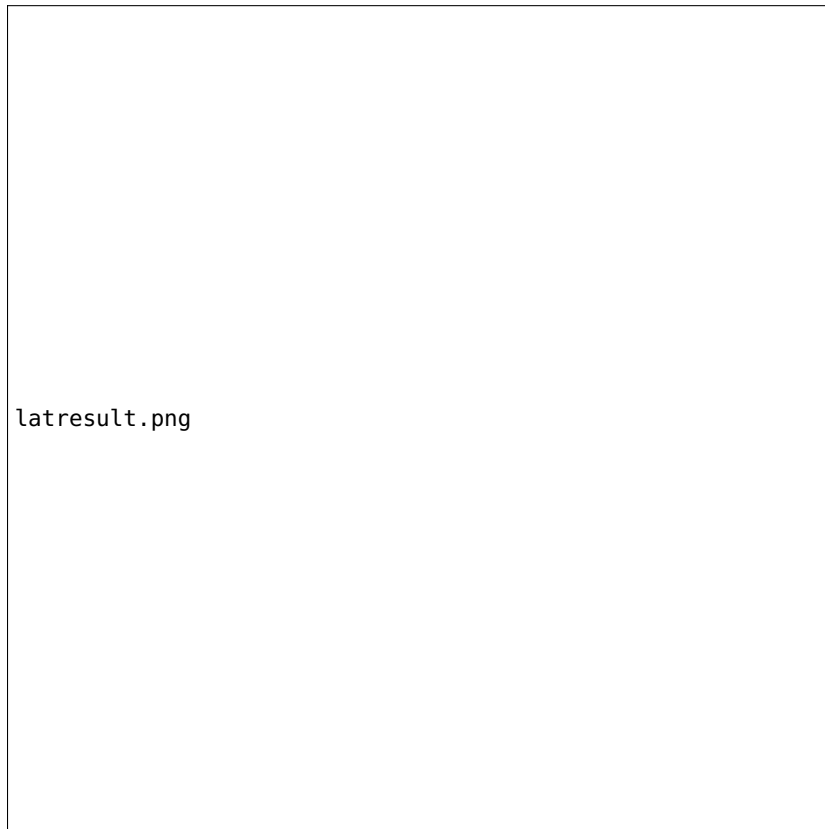
this is what a Latex paper can look like:

Figure 2: this is what latex paper can look like.

LaTeX

LaTeX is a document preparation system for high-quality typesetting. It is most often used for medium-to-large technical or scientific documents but it can be used for almost any form of publishing.

LaTeX is not a word processor! Instead, LaTeX encourages authors not to worry too much about the appearance of their documents but to concentrate on getting the right content.

LaTeX can be used for:

- Typesetting journal articles, technical reports, books, and slide presentations.

- Control over large documents containing sectioning, cross-references, tables and figures.

- Typesetting of complex mathematical formulas.

- Advanced typesetting of mathematics with AMS-LaTeX.

- Automatic generation of bibliographies and indexes.

- Multi-lingual typesetting.

- Inclusion of artwork, and process or spot colour.

- Using PostScript or Metafont fonts.

What the strings are:

In computer programming, a string is traditionally a sequence of characters, either as a literal constant or as some kind of variable. The latter may allow its elements to be mutated and the length changed, or it may be fixed (after creation). A string is generally understood as a data type and is often implemented as an array of bytes (or words) that stores a sequence of elements, typically characters, using some character encoding. A string may also denote more general arrays or other sequence (or list) data types and structures.

Python has a printf()-like facility to put together a string. The % operator takes a printf-type format string on the left (%d int, %s string, %f%g floating point), and the matching values in a tuple on the right (a tuple is made of values separated by commas, typically grouped inside parenthesis)

In case of def, it is the string that the word right behind the def len(a) is the length of (a) range is the limitation of the array from something to something. If there is a number of range, it starts from 0 to a-1 If there is (a,b), it starts from a to b-1 print gives the answer of the code and literally gives the answer return keep the answer for using again. import is for bringing libraries # is a comment for code that makes codes not to work

Common string operations

".

The string module contains a number of useful constants and classes, as well as some deprecated legacy functions that are also available as methods on strings. In addition, Python's built-in string classes support the sequence type methods described in the Sequence Types section, `str`, `unicode`, `list`, `tuple`, `bytearray`, `buffer`, `xrange` section, and also the string-specific methods described in the String Methods section. To output formatted strings use template strings or the `%` operator described in the String Formatting Operations section. Also, see the `re` module for string functions based on regular expressions

picture.png

This is string Source code mean .

Representing Python Code in Your Assignment

```
while b**e<c:
    x=a%(b**(e+1))
    y=x/(b**(e))
    ans=str(y)+ans
    a=a-x
    e=e+1
print ans
```

Here is some code can show you how to use the `str` in python.

3111

Integers and Floating Point Numbers

What is Integer?

An integer (from the Latin integer meaning "whole") [note 1] is a number that can be written without a fractional component. For example, 21, 4, 0, and -2048 are integers, while 9.75, $\frac{5}{2}$, and $\frac{1}{2}$ are not.

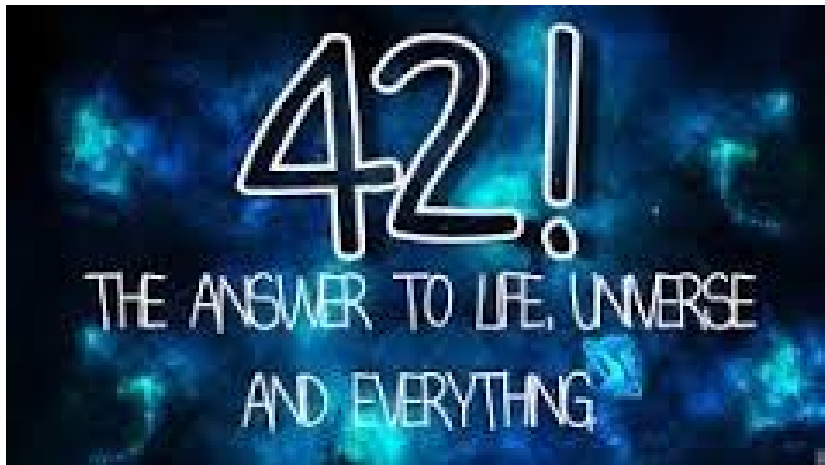


Figure 3: The ultimate answer of the universe is also an Integer!

[?]

Floating point!

Also called floats, they represent real numbers and are written with a decimal point dividing the integer and fractional parts. You can convert an integer to a floating number by your python and vice versa! :

Type `int(x)` to convert `x` to a plain integer.

```
Type: int(4.22222)
Result: 4
```

Type `float(x)` to convert `x` to a floating-point number.

```
type: float(1)
Result: 1.0
```

Number Type Conversion

Python converts numbers internally in an expression containing mixed types to a common type for evaluation. But sometimes, you need to coerce a number explicitly from one type to another to satisfy the requirements of an operator or function parameter.

Definition:

Booleans:

In Python we have the following terms (characters and phrases) for determining if something is "True" or "False." Logic on a computer is all about seeing if some combination of these characters and some variables is True at that point in the program.

and

or

not

!= (not equal)

== (equal)

>= (greater-than-equal)

<= (less-than-equal)

True

False

We use these characters to make the truth or not.

NOT:

not False = True

not True = False

OR:

True or False = True

True or True = True

False or True = True

False or False = False

AND:

True and False = False

True and True = True

False and True = False

False and False = False

NOT OR:

not (True or False) = False

not (True or True) = False

not (False or True) = False

not (False or False) = True

NOT AND:

not (True and False) = True

not (True and True) = False

not (False and True) = True

not (False and False) = True

!:=:

1 != 0 = True

1 != 1 = False

0 != 1 = True

```
0 != 0 = False
==:
1 == 0 = False
1 == 1 = True
0 == 1 = False
0 == 0 = True
```


The Boolean

Introduction

Boolean is a data type used in computer, there having two value and usually is "True" and "False". And Intended to represent the truth values of logic and Boolean algebra.

The Boolean Operators

Boolean Operators are some words like (AND, OR, NOT) used as connection to combine or remove words in a research , resulting in more focused and productive results. This should save time and effort by eliminating inappropriate hits that must be scanned before discarding, i thing these operators can greatly reduce or expand the amount of records returned.

Using these operators can reduce or expand the amount of records returned. Boolean operators are useful in saving time by focusing searches for more 'on-target' And now i would like to introduce the operators one by one with some examples.

AND – requires both terms to be in each item returned. If one term is contained in the document and the other is not, the item is not included in the resulting list. (Simpifly the search)

Example:

True AND True = True

True AND False = False

False AND True = False

False AND False = False

OR – either term (or both) will be in the returned document.
(Broadens the search)

Example:

```
True OR True = True
True OR False = True
False OR True = True
False OR False = False
```

NOT – first term is searched, then any records containing the term after the operators are subtracted from the results. The result needs to be the opposite.

Example:

```
NOT(True) = False
NOT(False) = True
```

Boolean Arithmetic

We can get booleans by making some statements like calculation symbols, and now we have " != ", " == ", " > ", " < ", " <= ", " >= ".

```
" != " symbol is means not EQUAL.
" == " symbol is what we call equal.
" > " symbol means greater than.
" < " symbol means less than.
" >= " symbol means greater and equal to.
" <= " symbol means less and equal to.
```

For example:

```
" == "
9==8 will get False.
9==9 will get True.
NOT(9==8) will get True.
NOT(9==9) will get False.
```

```
" > "
9>8 will get True.
8>9 will get False.
```

NOT(9>8) will get False.

NOT(8>9) will get True.

" <= "

9<=8 will get False.

8<=9 will get True.

8<=8 will get True.

NOT(9<=8) will get True.

NOT(8<=9) will get False.

NOT(8<=8) will get False.

Now we have some example for combination with symbols.And
.Lets try it together.

EXAMPLES:

NOT(9<7) AND (6>=7 OR 8==8)

(9*3>=35) OR NOT((77/11)==7) AND (6>=6 OR 54>4) AND (8<5 OR 33+3==10*4-4)

(9+2-4*2<=36/6) AND (8!=9 OR NOT(5*5-65/5==5))

NOT(6+4==7+3-5+4-1+24/8) OR ((6<3*3)!=(4+4+5-3))

Boolean Operations and, or, not

1.OR

This is a short-circuit operator, so it only evaluates the second argument if the first one is False.

2.And

This is a short-circuit operator, so it only evaluates the second argument if the first one is True.

3.Not

not has a lower priority than non-Boolean operators, so not a == b is interpreted as not (a == b), and a == not b is a syntax error

Comparisons

Comparison operations are supported by all objects. They all have the same priority (which is higher than that of the Boolean operations). Comparisons can be chained arbitrarily; for example, $x < y \leq z$ is equivalent to $x < y$ and $y \leq z$, except that y is evaluated only once (but in both cases z is not evaluated at all when $x < y$ is found to be false).

This table summarizes the comparison operations

a)equal"=="

use == check the the date is equal or not. example 3 == 7 is false.

b)strictly greater than ">"

c)greater than or equal">="

d)strictly less than"<"

e)less than or equal"<="

Conditional Statements

Introduction

I am going to talk about if and else. The if statement is used for conditional execution. It selects exactly one of the suites by evaluating the expressions one by one until one is found to be true. if all of the expressions are false, it will choose else. And the elif statement. It stands for "else if," which means that if the original if statement is false and the elif statement is true. An else statement can be combined with an if statement. An else statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a false value.

```
a=15
b=10
if a>b:
    print "a>b"
else: print "a<=b"
```

And the output:

a>b

if "a" greater than "b", print "a>b". For example, "a" is 15, "b" is 10, 15 is greater than 10, so print "a>b". if "a" is not greater than "b", then, choice else, print "a<=b"

```
a = 0
```

```
while a < 10:
    a = a + 1
    if a > 5:
        print (a, ">", 5)
    elif a <= 7:
        print (a, "<=", 7)
    else:
        print ("Neither test was true")
```

And the output:

```
1 <= 7
2 <= 7
3 <= 7
4 <= 7
5 <= 7
6 > 5
7 > 5
8 > 5
9 > 5
10 > 5
```

first,because of $a < 10$. $a = 1 + 1$ the result should greater than 5 so "a" will from 0 to 9 until a cannot change any more, so, $a = 6, 7, 8, 9, 10$. if not, like $a < 7$ or $a = 7$, print $a <= 7$. so 1,2,3,4,5 is the answer.

In the Game of Life we updated the grid depending on the number of neighbors of a cell.

```
if A[i][j]==0:
    if neigh==3:
        B[i][j]=1
    else:
        B[i][j]=0
else:
    if neigh==2 or neigh==3:
        B[i][j]=1
    else:
        B[i][j]=0
```

While Loops

A **while loop** statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

The condition may be any expression, and true is any non-zero value. The loop iterates while the condition is true. When the condition becomes false, program control passes to the line immediately following the loop. In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

Python code example

```
count = 0
while (count < 9):
    print 'The count is:', count
    count = count + 1

print "Good bye!"
```

Output

```
>>>
The count is: 0
The count is: 1
The count is: 2
The count is: 3
```

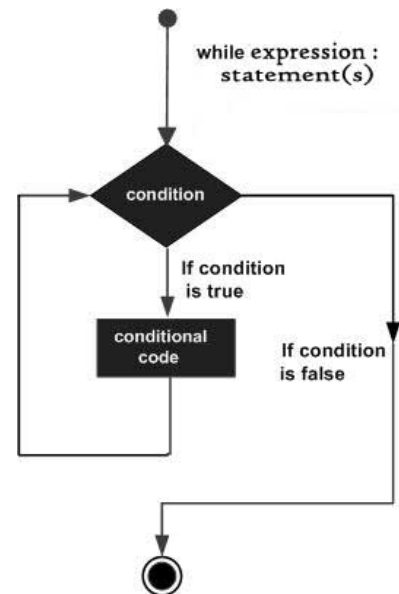


Figure 4: Flow diagram about how the while loop works

The code will produce the following output.

```

The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
>>>

```

Infinite Loop

A loop becomes **infinite loop** if a condition never becomes **FALSE**. You must use caution when using while loops because of the possibility that this condition never resolves to a FALSE value. This results in a loop that never ends. Such a loop is called an infinite loop.

An infinite loop might be useful in client/server programming where the server needs to run continuously so that client programs can communicate with it as and when required.

```

var = 1
while var == 1:
    num = raw_input("Enter a number :")
    print "You entered: ", num

print "Good bye!"

```

This python code is an example of how infinite loop can be created.

```

Enter a number :X
You entered:  x
Enter a number :Y
You entered:  Y
Enter a number :Z
You entered:  Z
Enter a number between :

```

This code creates an infinite loop where it will need your input of any number. Once you input any number, it will output it like if you input "X" it will show back "X".

To break the loop you will either need to add the "**break**" command in your code OR press **CTRL+C** to exit the program.

Using else statements with while loops

Python supports to have an else statement associated with a loop statement.

If the **else statement** is used with a while loop, the else statement is executed when the condition becomes false.

```
count = 0
while count < 5:
    print count, " is less than 5"
    count = count + 1
else:
    print count, " is not less than 5"
```

The following code illustrates the combination of an else statement with a while statement that prints a number as long as it is less than 5, otherwise else statement gets executed.

```
>>>
0 is less than 5
1 is less than 5
2 is less than 5
3 is less than 5
4 is less than 5
5 is not less than 5
>>>
```

This is the output in python when the code above is executed.

While Loops

A while loop is a function, which needs a boolean statement to run, in order to print out a list of results. The while loop will print out as many results as possible, until the boolean statement stops being true. For example:

```
i=0
while (i<5):
    print i
    i=i+1
```

Now, the boolean statement inserted here is "i<5", meaning that "i" must be less than 5. The next function now commands the system to print "i". The result would be:

```
0
1
2
3
4
```

This is because after the computer was demanded to print out "i", the function "i==i+1", was entered, meaning that it should print out all the numbers that make the boolean statement true, until the number is five, making it false.

There are also many ways you could manipulate this code. A break may be inserted as follows:

```
i=0
while (i<5):
    print i
    i=i+
    if i==4:
        break
```

The result would be:

0
1
2
3

Excluding "4", because as a result of the break, the computer is now told not to print anything from the number 4.

However, if the break came before "i=i+1" like:

```
i=0
while (i<5):
    print i
    if i==4:
        break
    i=i+1
```

The result would be:

0
1
2
3
4

Bases

The word "base" in mathematics is used to refer to a particular mathematical object that is used as a building block. The most common uses are the related concepts of the number system whose digits are used to represent numbers. In common, people use decimal system. However, the computers are composed with binary system. Also, SHA-256 gives the hexadecimal code.

Binary

Binary is composed with only 0 and 1. Each of digits represents 2 to the power of something. From the right to left, it starts from 2^0 to increasing the power.

Decimals

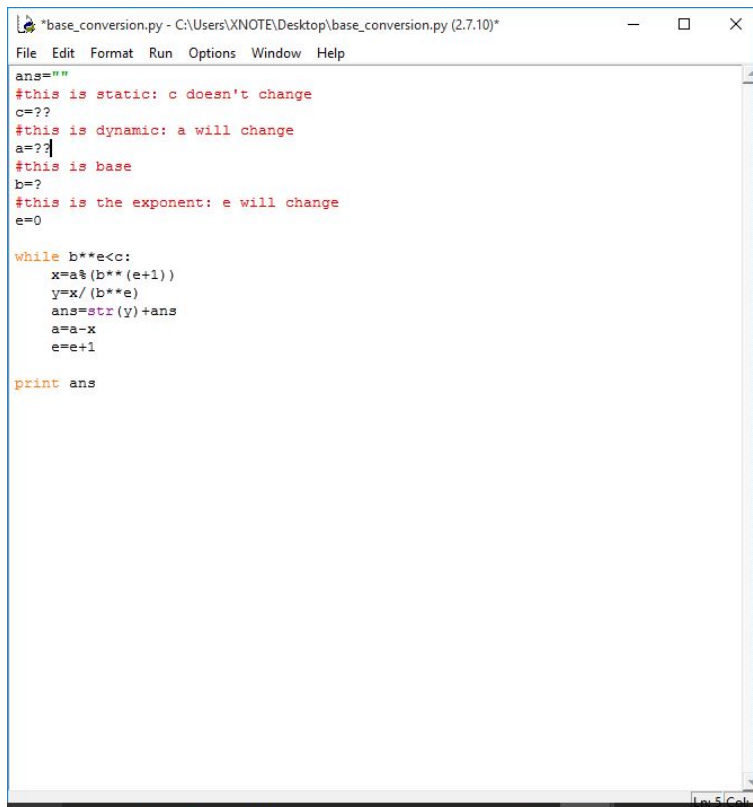
Decimal system is what people use in usual days. It is composed with 0,1,2,3,4,5,6,7,8, and 9, and it starts from 10^0 and increases the power of 10 when it goes the next digit. The reason why people use the decimal system is that the human has 10 fingers, so the calculation is easier and more simple.

Hexadecimals

Hexadecimal system is used for computer languages such as C language/C++ and SHA-256. It is composed with 1,2,3,4,5,6,7,8,9,a,b,c,d,e, and f. When the decimal numbers are hashed by SHA-256, it is more difficult if hashed number starts with 0 and have many 0s in the number. The number in front of x is the number of 0s which is leading.

Conversion

It is the sample code of the base conversion. ? is what the base is, so it can be any decimal number. ?? can also be the any number,



```
*base_conversion.py - C:\Users\XNOTE\Desktop\base_conversion.py (2.7.10)*
File Edit Format Run Options Window Help

ans=""
#this is static: c doesn't change
c=?
#this is dynamic: a will change
a=?
#this is base
b=?
#this is the exponent: e will change
e=0

while b**e<c:
    x=a%(b**(e+1))
    y=x/(b**e)
    ans=str(y)+ans
    a=a-x
    e=e+1

print ans
```

but decimal numbers. It changes from decimal numbers to base of ? numbers. a needs to be subtracted by x for the next step getting ? to the power of next exponent. Then, repeat the previous steps with e added 1. And then, repeat for next step again. At the last, however, y which is the quotient of the last x divided b to the power of e should be concatenated with ans . The quotient becomes the first number of the ? base numbers.

Array 1-D, 2-D, 3-D

Intro

Array is like a storage, it can fill with string or integer. In 1-D, 2-D it can also represents the x-axis and y axis.

Creating arrays

Arrays is created buy blanket.

Example:

```
a[ ]      *a is the array name that you want.
```

The things in the [] and be store and when you want to access it you will need its position in the array and type like a[o]

Example:

```
a=["apple","orange","banana"]
```

If you want to print banana form the array, you may want to type

```
print a[2]
```

Filling arrays

Everything can be store in the array, strings, integers, arrays. When you create an array you can fill things in it as the default things that the array have.

Example:

```
a["Billy","Bud",90,60,50]  
b["Anne","Chow",90,95,100]  
c["Jen","Bo",60,80,90]
```

If you want to add things into the array that u create already, you can use

```
array_name=array_name+[Things you want to add]
```

Example:

```
a=[apple]
```

and now I want to add orange into it, so we add

```
a=a+[orange]
```

To create 2-D or more array we need to create array in the nested for-loop.

Example:

```
a=[]
for i in range(N): *N how long you want the array to be
    b=[]          *This is a temporary array to generate every array inside the main array.
    for j in range(N):
        *Things you want to put in the array by b=b+[ ]
    a=a+[b]        *Here put the temporary array back to the main array.
```

Traversing array

Traversing array is visiting each element in the array and do something. In 1-D we can do it with for loop to identify things in array.

Example:

```
a=[1,2,3]
for i in range(len(a))    *len(a) = Numbers of elements in the array
    *Things put here can edit the specific element a[i]
```

In 2-D we start using nested for-loop to identify the x-axis and y-axis. So we use nested for-loop to traversing it too.

Example:

```
a=[[0,1],[0,0],[0,1]]
for i in range(len(a)):
    for j in range(len(a)):
        *Things put here can edit the specific element a[i][j]
```

In 3-D we use more for-loop to identify the more dimension.

Example:

```
a=[[[0,0],[0,0]],[[0,0],[0,0]]]
for x in range(len(a)):
    for y in range(len(a)):
        for z in range(len(a)):
            *Things put here can edit the specific element a[x][y][z]
```


Arrays

In a nutshell, arrays are lists of data. Most commonly, they are sets of integers, although they can also be sets of strings. There are many uses for an array, such as being used as a database to count occurrences. The simplest arrays are one-dimensional arrays, which look like this.

```
[0,1,2,3,4]
["a","b","c","d","e"]
```

These arrays can be described with several different characteristics.

1. They are both an array of length 5.
2. They are both one dimensional arrays (trust me, this will make more sense).
3. The first is a list of integers, and the second is a list of strings.

Array generation

Arrays are generated in a program by first creating an empty array, or by creating an array that is already filled with whatever you'd like.

```
a = []
savoryfillings = ["meat","cheese","potato"]
```

Adding to an array

Adding to an array, also known as “concatenation”, is also a very simple process, much like generating an array. A simple way to fill an array with, let's say, zeroes, is with a for loop.

```
a = []
for i in range(5):
    a = a + [0]
print a
#this program will yield [0, 0, 0, 0, 0]
```

Note how the zero is in brackets. This will signify to the interpreter that this zero is intended to be a unit of the array.

Strings can also be concatenated to an array of strings.

```
>>> savoryfillings = ["meat","cheese","potato"]
>>> savoryfillings = savoryfillings + ["spinach"]
>>> print savoryfillings
['meat', 'cheese', 'potato', 'spinach']
```

Referencing certain parts of an array

Referencing certain parts of an array is rather simple. A simple program that does such thing and prints the referenced points is displayed below. Remember that Python starts counting at 0, not 1!

```
>>> a = [1, 2, 3, 4]
>>> x = a[0]
>>> y = a[1]
>>> #referencing points of array is done by (nameofarray)[pointinarray]
>>> print x
1
>>> print y
2
```

The command 'len'

The 'len' command takes the length of a given array and converts it into an integer.

```
>>> a = [1, 2, 3]
>>> x = len(a)
>>> print x
3
```

'len' can also be used to do something to every point of an array, with the use of a for loop.

```
>>> a = [0, 1, 2, 3, 4, 5]
>>> for i in range(len(a)):
a[i] = a[i] + 1

>>> print a
[1, 2, 3, 4, 5, 6]
```

Two-/three-dimensional arrays

Multidimensional arrays can be created through the use of nested for loops - for example, the program below generates a 5x5 array of zeroes.

```
>>> a = []
>>> for i in range(5):
    b = []
    for j in range(5):
        b = b+[0]
    a = a+[b]
>>> print a
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
```

Note how this two-dimensional array has five arrays within it, each with five zeroes.

Application of arrays

Arrays can be used to catalog data. This program logs

```
import random
mastercount = []
for i in range (10):
    mastercount = mastercount + [0]
for i in range (1000):
    x = random.randint(0,9)
    mastercount[x] = mastercount[x]+1
print mastercount
#sample yield of program - [95, 103, 96, 98, 107, 108, 101, 107, 102, 83]
```


For Loops, Nested For Loops

For in loop has the ability to iterate over the items of any sequence, such as a list or a string.

If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable iterating var. Next, the statements block is executed. Each item in the list is assigned to iterating var, and the statement(s) block is executed until the entire sequence is exhausted.

PYTHON CODE

Syntax:

```
for var in sequence:  
    statements
```

Example 1

```
for i in ["hello", "hey", "yo"]:  
    print i
```

The goal is to print all the elements in the array

Output 1

```
>>> Hello  
      hey  
      yo
```

Example 2

```
for i in range(5):  
    print i
```

The goal is to print all the number in range 5. Note, that Python starts counting from 0

Output 2

```
>>> 1  
    2  
    3  
    4  
    5
```

Nested for loops

Python programming language allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.

Syntax

```
for iterating_var in sequence:  
    for iterating_var in sequence:  
        statements(s)  
    statements(s)
```

Example 1

```
for i in range(5): #loop everything indented 5 times  
    for j in range(3): #loop everything indented 3 times  
        print i #print output of i
```

The goal is to print every number from 0 to 5 three times in a row.

Output 1

```
>>> 0
      0
      0
      1
      1
      1
      2
      2
      2
      3
      3
      3
      4
      4
      4
```

Example 2

```
import random #importing "random" library that will take random numbers for our program
x = [] #creating initial, empty array that will be fulfilled with other arrays later
for i in range(5): #loop everything indented 5 times
    r=[] #creates 5 more arrays
    for j in range(5): #loop everything indented 5 times
        if random.uniform(0,10)<3: #compare if randomly chosen number is smaller than 3
            r=r+[1] #if it is, add value of "1" inside the secondary array - r
        else: #or
            r=r+[0] #if it is bigger, add value of "0" inside the secondary array - r
    x.append(r) #add all of these secondary arrays to our main - x array.
#Basically we create 2D array.
print x #print our 2D array to see the output.
```

The goal is to create a 2D arrays that would be fulfilled with values of "1" or "0" depending on which number was randomly chosen by the computer

Output 2

```
>>> [[0, 0, 0, 1, 1], [1, 0, 0, 0, 0], [0, 0, 0, 0, 0],
      [0, 1, 0, 0, 0], [0, 0, 0, 0, 0]]
```

However, the numbers that you see here isn't the only output you can get. We used random uniforms and each time it will give different values.

Introduction–python functions

python has lots of pre-made functions , that you can use right now, simply by "calling" them. The concept of a function is one of the most important ones in mathematics. A common usage of functions in computer languages is to implement mathematical functions. Functions are a convenient way to divide your code into useful blocks, allowing us to order our code, make it more readable, reuse it and save some time. Also functions are a key way to define interfaces so programmers can share their code.

Defining a Function

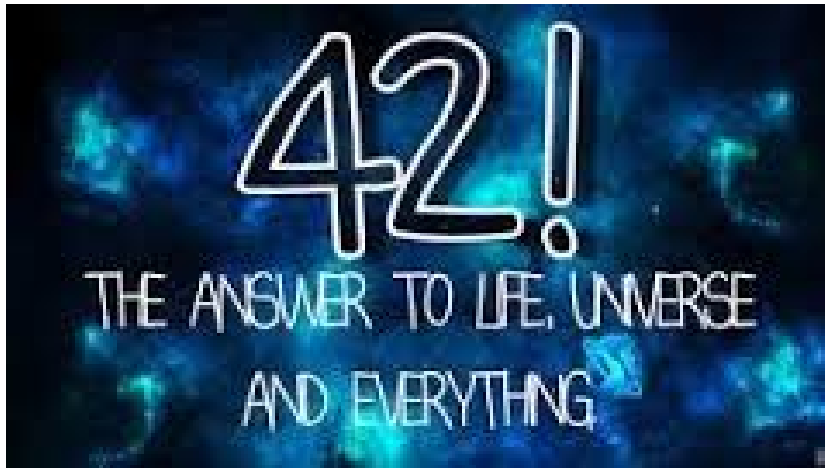


Figure 5: The Universe

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

Function blocks begin with the keyword `def` followed by the function name and parentheses `()`.

Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.

The first statement of a function can be an optional statement - the documentation string of the function or docstring.

The code block within every function starts with a colon `(:)` and is indented.

The statement `return [expression]` exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as `return None`.

Generator Functions

Generators are a simple and powerful possibility to create or to generate iterators. On the surface they look like functions, but there is both a syntactical and a semantic difference. Instead of return statements you will find inside of the body of a generator only yield statements, i.e. one or more yield statements.

The following is a simple example of a generator, which is capable of producing four city names:

```
def city_generator():
    yield("Konstanz")
    yield("Zurich")
    yield("Schaffhausen")
    yield("Stuttgart")
```

It's possible to create an iterator with this generator, which generates one after the other the four cities Konstanz, Zurich, Schaffhausen and Stuttgart.

```
>>> from city_generator import city_generator
>>> x = city_generator()
>>> print x.next()
Konstanz
>>> print x.next()
Zurich
>>> print x.next()
Schaffhausen
>>> print x.next()
Stuttgart
>>> print x.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
>>>
```


Libraries

Introduction

Libraries in Python are extensions to the basic Python coding. Python comes with some libraries of its own. But it is also possible to write your own libraries. But before you can use libraries you have to import the libraries that you want to use in your script. There are several ways how you can import libraries. Libraries are always imported at the beginning of a script.

Importing Python Libraries

The easiest way to import libraries is to use the import function. For these examples we will use the random library.

With this you have to put the libraries name in front of the function of the library

```
1 import random
2 print random.uniform(1,10)
```

If you want to rename a library before you are using it you can do the following

```
1 import random as rndm
2 print rndm.uniform(1,10)
```

If you don't want to have to write a library name in front of it at all you can do

```
1 from random import *
2 print uniform(1,10)
```

There is one other option how you can import libraries. If you use all of what we learned before we can use the following.

With this you can import a single function of a library and you name the function.

```
1 from random import uniform as makeRandom
2 print makeRandom(1,10)
```

Importing custom libraries

If you want to use libraries you or someone else has written in python you can do that also. First you have to make sure that the script you want to import is in the same folder as the script you want to import it into. Lets assume we have the following script we want to use as a library.

```
1 def Bla():  
2     print "Bla"  
3  
4 def MyFunction(A)  
5     print A+A
```

Lets assume the script's name is lib. Now if we want to use this in our main script we can do either of our ways. Use the file name as the library's name.

```
1 import lib  
2 import lib as mylib  
3 from lib import Bla as Tell
```


Random

Intro

Random is a library that you can use to generate random numbers in python.

Creating random numbers

You can use a code to generate random numbers.

```
import random    *You need to import random before you generate random numbers
random.uniform(minimum,maximum)
```

while minimum and maximum number can be intergers(o) or floats(o.o)

Usage of random number

Simply here is a example of how to add random numbers to an array

```
import random
a=[]
for i in range(N):    *N is the the number of how many random numbers you want to add to the array
    a=a+[random.uniform(0,100)]
    *0 is the minimum number and 100 is the maximum number, they are variable, you can change to
    numbers you want.
```

Here is a example of using random function to randomly fill a 2-D array with 0 and 1 by how many percentage you want it is filled.

```
import random
def fill(a,p):    *a is the name or you 2-D array    *p is the percentage you want, such as: 0.1, 0.25....
    for i in range(len(a)):    *len(a) is the length of you array
        for j in range(len(a)):
            if random.uniform(0,1)<p:
                a[i][j]=1
            else:
                a[i][j]=0
```


Statistics Library

Introduction

This statistics library includes eight functions that we can use to deal with a set of data.

Tips

- *Additional libraries are needed*
- *Multiple functions are needed for some operations*

Zero

```
def zeros(n):  
    a=[]  
    for i in range(n):  
        a=a+[0]  
    return a
```

(1)

This function can be used to create a consecutive and repeating array(all elements are the same).

Example:

```
def zero(n):  
    a=[]  
    for i in range(n):  
        a=a+[0]  
    return a
```

```
print zero(9)
```

Output: [0, 0, 0, 0, 0, 0, 0, 0, 0]

Summing an array

```
def sum_array(a):
    s=0
    for i in range(len(a)):
        s=s+a[i]
    return s
```

(2)

This function can help us to sum up all the elements in an array.

Example:

```
a=[1,2,3,4]
def sum_array(a):
    s=0
    for i in range(len(a)):
        s=s+a[i]
    return s
print sum_array(a)
```

Output: 10

Finding means

```
def avg(a):
    return sum_array(a)/len(a)
```

(3)

This function can help us to calculate the mean of all the data in an array, and we need assistance of the sum function.

Example:

```
a=[23,19,25,21]
def sum_array(a):
    s=0
    for i in range(len(a)):
        s=s+a[i]
    return s

def avg(a):
    return sum_array(a)/len(a)
```

```
print avg(a)
```

Output: 22

Variance

```
def var(a):
    s=0
    for i in range(len(a)):
        s=s+a[i]**2
    m=avg(a)
    return (s/len(a)-m**2)
```

(4)

This function can help us to find the variance of the data. Also, we need the assistance of the average function.

Example:

```
a=[23,19,25,21]
def sum_array(a):
    s=0
    for i in range(len(a)):
        s=s+a[i]
    return s

def avg(a):
    return sum_array(a)/len(a)

print avg(a)

def var(a):
    s=0
    for i in range(len(a)):
        s=s+a[i]**2
    m=avg(a)
    return (s/len(a)-m**2)
print var(a)
```

Output: 5

Construct an array of random numbers

```
def rand_array(n,mini,maxi):
    a=[]
    for i in range(n):
        a=a+[random.uniform(mini,maxi)]
    return a
```

(5)

We can use this function to construct an array filled with random numbers. We need the random library to run the function. Here n represents the number of elements in the array; $mini$ is the minimum value of the elements; $maxi$ is the maximum value

To fill a histogram

*Library "graphics" is needed**

```
def histogram(mini,maxi,bins,a):
    h=zeros(bins)
    w=(maxi-mini)/bins
    for i in range(len(a)):
        for j in range(bins):
            if (a[i]>(mini+j*w))and a[i]<(mini+(j+1)*w):
                h[j]=h[j]+1
    return h
```

(6)

The four arguments in parenthesis are decisive for the histogram. Here $mini$ represents the minimum value in the data; $maxi$ represents the maximum value in the data; bin represents the number of different groups of data; a is the number of all the data.

Find the maximum value

```
def maximum(a):
    m=0
    for i in range(len(a)):
        if a[i]>m:
            m=a[i]
    return m
```

(7)

This function can help us to identify the maximum value in the data.

Drawing a histogram

Library "graphics" is needed

To draw a histogram, we first need to define a function that can draw a bargraph:

```

def bar(a,win):
    win.setCoords(-1,-1,len(a)+1,maximum(a)+1)
    bl=[]
    tr=[]
    rec=[]
    for i in range(len(a)):
        bl=bl+[Point(i,0)]
        tr=tr+[Point(i+a,a[i])]
        rec=rec+[Rectangle(bl[i],tr[i])]
        rec[i].draw(win)

```

(8)

Then we combine the "bar" function and the "histogram" function.
Here is just an example:

```

def main():
    win=GraphWin()
    a=zeros(400)
    for i in range(len(a)):
        a[i]=sum_array(rand_array(10,0,1))
    histo=histogram(0.,10.,7,a)
    bar(histo,win)
main()

```

(9)

Statistics

Introduction

Statistics is the study of the collection, analysis, interpretation, presentation, and organization of data.[1] In applying statistics to, e.g., a scientific, industrial, or societal problem, it is conventional to begin with a statistical population or a statistical model process to be studied. Populations can be diverse topics such as "all persons living in a country" or "every atom composing a crystal". Statistics deals with all aspects of data including the planning of data collection in terms of the design of surveys and experiments.

Statistics library

```
import random

def zeros(n):
    a=[]
    for i in range(n):
        a=a+[0]
    return a
def sum_array(a):
    s=0
    for i in range(len(a)):
        s=s+a[i]
    return s
def rand_array(n,mini,maxi):
    a=[]
    for i in range(n):
        a=a+[random.uniform(mini,maxi)]
    return a
```

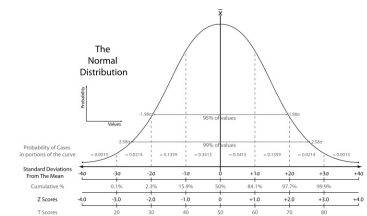


Figure 6: More probability density is found as one gets closer to the expected (mean) value in a normal distribution. Statistics used in standardized testing assessment are shown. The scales include standard deviations, cumulative percentages, percentile equivalents, Z-scores, T-scores, standard nines, and percentages in standard nines..

In the begin, I need type in "import random" to start this statistics code .

The first code in statistics library is the explanation of the zero, it is a basic mode to calculate.

The second code is to define the sum of array, it will help user to calculate the sum when the array is longer than affordability.

The third code is to define the rand of array, this code is start more complex than before, we need the n, mini and maxi to finish the calculation

```

def avg(a):
    return sum_array(a)/len(a)
def var(a):
    s=0
    for i in range(len(a)):
        s=s+[a[i]**2]
    m=avg(a)
    return (s/len(a)-m**2)
def maximum(a):
    m=-math.inf
    for i in range(len(a)):
        if a[i]>m:
            m=a[i]
    return m

```

The fourth code is to know the average of the array, we will use the sum of array divide by length of array to calculate this, and we already know how to type the sum of array, so it is not a no clue function.

In the fifth code, the var is when you define a function parameter and it is a variable number of parameters of meaning, so we can use this code to figure out it very convenient.

```

def histogram(mini,maxi,bins,a):
    h=zeros(bins)
    w=(maxi-mini)/bins
    for i in range(len(a)):
        for j in range(bins):
            if a[i]>(mini+j*w) and a[i]<(mini+(j+1)*w):
                h[j]=h[j]+1
    return h
def bargraph(a):
    win=graphWin("BarGraph",400,400)
    win.SetCoords(-1,-1,len(a)+1,maximum(a)+1)
    for i in range(len(a)):
        rec=Rectangle(Point(i,0),Point(i+1,a[i]))
        rec.draw(win)
def main():
    gauss=[]
    for i in range(100):
        gauss=gauss+[sum_array(rand_array(10,0,1))]
    bargraph(histogram(0,10,10,gauss))

```

The sixth code is to calculate the maximum.

The seventh code is the histogram, we need use the value of mini,maxi,bins and a.

The last two code is bargraph and main, be honest, those two code is relatively strange for me, but in the statistics library, it will be a good supporting code to reduce your time for anytime you need to use those code.

Graphics Library

The graphics library is a library for making easy graphical objects in python. It was written by John Zelle for use with his book Python Programming: An Introduction to Computer Science. This chapter is going to discuss the topics we covered during class. To use the library you have to import it first. For an explanation on how to import libraries see the section of the book.

Creating a basic window

The graphics library works as a program of its own. A window gets updated as long as it gets changed or is waiting for a new command. To draw something on a window we first have to create a window. A creation follows this pattern:

```
1 Window = GraphWin(WindowName, Width, Height)
2 Window.setCoords(StartX, StartY, WidthPixel, HeightPixel)
```

Drawing on windows

Now that we know how to create a window we can start drawing on it. There are various objects we can draw on windows. Because the graphics library uses its own coordinates system we need to use it for creating objects.

```
1 Point(2,4) #Creates a point at 2,4
```

To actually draw in windows we have to use the draw command.

```
1 Window = GraphWin(WindowName, Width, Height)
2 Window.setCoords(StartX, StartY, WidthPixel, HeightPixel)
3 Obj = OurObject
4 Obj.draw(Window)
```

In this example we are drawing Obj to our window.

Rectangles

In python rectangles are drawn from the down left point to the top right point.

```
1 Rect = Rectangle(Point(X1,Y1),Point(X2,Y2))
```

This creates a rectangle calles Rect at the position X1 and Y1 with the width X2 and height Y2. After you have created your basic reactangle you can also color it.

```
1 Rect = Rectangle(Point(X1,Y1),Point(X2,Y2))
2 Rect.setFill(color_rgb(255,0,0)) #Set the color for the reactangle
```

Circles

Of course you can also add circles to your window. Circles are drawn from their middle point to the setting of the size of their diameter.

```
1 C = Circle(Point(X,Y),Point(RX,RY))
```

The Point(X,Y) is defining the position of the circle. The Point(RX,RY) is defining the radius of the circle.

Text labels

On a window it is very important to be able to use text. Because of that the graphics library also comes with a text object for windows.

```
1 T = Text(Point(X,Y),StringText)
```

The example is self explaining. But this text is nothing special. To make it cool we need to modifie it a bit.

```
1 Text.setFace(Font)
2 #Availible Fonts are 'helvetica','arial','courier','times roman'
3 Text.setSize(Size) #Size of the text
4 Text.setStyle(Style)
5 #Availible styles 'bold','normal','italic', 'bold italic'
6 Text.setTextColor(Color) # Use color_rgb(r,g,b) as a color
```

Other usefull methods

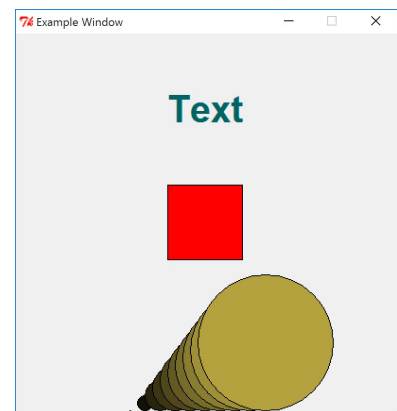
There are some other usefull methods for the graphics library. Obj can be any object of the graphics library.

```
1 Obj.setFill(Color) #Sets the color
2 Obj.clone() #Creates a new object with the same properties as Obj.
3 Obj.move(X,Y) #Moves the object
4 color_rgb(r,g,b) #Makes a color for use with this library
5 Window.getMouse() #Stops the programm until you press on the screen
```

Example code

This is an example code of the graphics library.

```
1 from graphics import *
2
3 Win = GraphWin("Example Window",400,400)
4 Win.setCoords(0,0,50,50)
5 T = Text(Point(25,40), "Text")
6 T.draw(Win)
7 T.setFill(color_rgb(0,100,100))
8 T.setStyle("bold")
9 T.setFace("arial")
10 T.setSize(30)
11
12 Rect = Rectangle(Point(20,20),Point(30,30))
13 Rect.draw(Win)
14 Rect.setFill(color_rgb(255,0,0))
15
16 for i in range(10):
17     C = Circle(Point(15+i*2,i),i)
18     C.draw(Win)
19     C.setFill(color_rgb(20*i,18*i,7*i))
```



Graphics

GRAPHICS IN PYTHON IS AN AMAZING TOOL FOR CREATING A DOCUMENT. FOR INSTANCE, WHEN PEOPLE WANT TO SHOW A COMPARISON BETWEEN SOME SPECIFIC STATUSES, THEY MIGHT WANT TO CREATE A LINE DIAGRAM OR A PIE DIAGRAM. IN THIS CASE, THEY MIGHT USE PYTHON GRAPHICS TO GRAPH THE DIAGRAMS. FURTHERMORE, PEOPLE CAN ALSO USE GRAPHICS TO CREATE ANIMATION. FOR INSTANCE, IN CLASS, THE TEACHER TAUGHT US HOW TO CREATE A CONWAY GAME OF LIFE AND THE EXACT WAY TO DO IT IS TO USING CIRCLES AND SOME OTHER CODES TO COMPILE THE ANIMATION. IN SOME OTHER CIRCUMSTANCES, PEOPLE ALSO USE GRAPHICS TO CREATE GAMES AND OTHER THINGS. THE MOST FAMOUS EXAMPLE IS THE PAC MAN, WHICH HAS BASIC ELEMENTS AS CIRCLES AND SQUARES. AS THE RESULT, GRAPHICS IS ESSENTIAL FOR COMPUTER SCIENCE AND IT IS EXTREMELY USEFUL AS WELL.

-Example of Graphics

Here come some basic codes of compiling some shapes. For instance, people can compile a triangle, a square, a circle, a series of circle. Consequently, I am going to show some of the codes for compiling Squares, Circles, Windows, Lines, and texts.

(1)Bar graphics

```
from graphics import*
import random
#length of the array
L=10
#Max value of number
```



Figure 7: This is an example of python graphics.

First, open a new file and import the library called graphics. Second, import random number because we want random height of squares. After that, we set the length of array 10 and the max value of number to be 10 as well because we don't want the graph to be over flow. Third, we create an array in order to create random numbers. Forth, we create a window and find the bottom left points and top right points of the squares. Consequently, we can graph the squares. At the very end we are compiling the squares to the window that we created. Rule of creating a square, square=[Point(bottom left, top right)]



Figure 8: This is an example of python graphics, Squares.

```

M=10
a=[]
for i in range(10):
    a=a+[random.uniform(0,M)]

win=GraphWin()
win.setCoords(-0.2,0,L,M+0.2)

#array of bottom left points
bl=[]
for i in range(L):
    bl=bl+[Point(i,0)]

#array of top right points
tr=[]
for i in range(L):
    tr=tr+[Point(i+1,a[i])]

#array of rectangles
rec=[]
for i in range(L):
    rec=rec+[Rectangle(bl[i],tr[i])]

for i in range(L):
    rec[i].draw(win)

```

(2) 10 Circles

```

from graphics import *

wind=GraphWin()
wind.setCoords(0,0,10,12)

centers=[]
for i in range(10):
    centers=centers+[Point(5,1+i)]

circles=[]
for i in range(10):

```

Same as creating squares, import the library and create a new window. After that, we need to create an array for finding the center point of the 10 circles. Following, we follow the rules if creating circles which is `Circle(center point, radius)`. At the end, we compile the circles to the window.

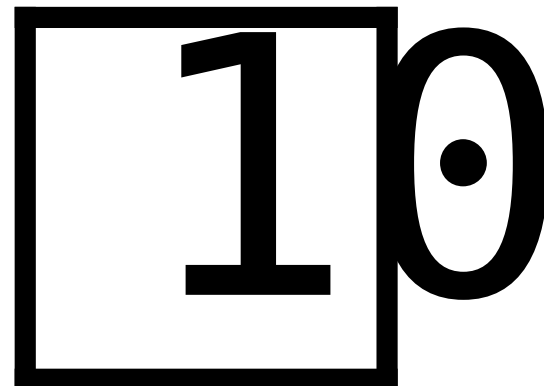


Figure 9: This is an example of python graphics, 10 circles.


```

        circles=circles+[Circle(centers[i],1)]

for i in range(10):
    circles[i].draw(wind)

```

(3)Window

```

from graphics import *

wind=GraphWin()
wind.setCoords(0,0,10,10)

```

In order to create a window, we also need to import graphics library. After that, we need to follow the rule of creating a window which is set the name of the window(in this case is wind). Following, we set the coordinate of the window which is wind.setCoords(0,0,10,10). This process creates a window for us to put other graphics in.

(4)Lines

```

from graphics import *

wind=GraphWin()
wind.setCoords(0,0,10,10)

Line= Line(Point(1.0,3.0),Point(1.0,4.0))

Line.draw(wind)

```

Same thing as other graphics, we import the library and create a window. After, we follow the rule of creating the line which is finding its starting point and the ending point. For instance, Line[Point(1.0,3.0),Point(1.0,4.0)]. At the end, we draw it on the window that we just created.

(5)text

```

from graphics import *

wind=GraphWin()
wind.setCoords(0,0,10,10)

text= Text(Point(0.0,5.0),"Hello")

text.draw(wind)

```

Creating text in graphics is also a simple process. First, we import the library and create a window for it. After that, we need to follow the rule of compiling text on a window which is Text(Point(coordinate), "text")

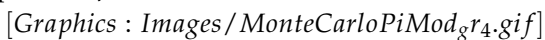
Monte Carlo

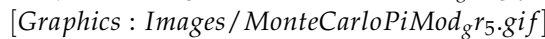
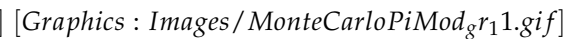
Monte Carlo simulations are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. They are often used in physical and mathematical problems and are most useful when it is difficult or impossible to use other mathematical methods. Monte Carlo methods are mainly used in three distinct problem classes:[1] optimization, numerical integration, and generating draws from a probability distribution. In Class we have used the Monte Carlo simulation for the probability function.

We have estimated the value of pi

We start the familiar example of finding the area of a circle. Figure 1 below shows a circle with radius $r=1$ inscribed within a square.

The area of the circle is $Pi * r^2 = Pi * 1 = Pi$ and the area of the square is 4 The ratio of the area of the circle to the area of the square

is 

```
import random
x=random.uniform(-1,1)
y=random.uniform(-1,1)
n=0.
#n is the number of random points in the circle
p=999999
#p is the number of random prints
for i in range(p):
    x=random.uniform(-1,1)
    y=random.uniform(-1,1)
    if (x*x+y*y)<1:
        n=n+1
pi=4*(n/p)
print pi
```

As you increase P the estimation will be more accurate. for p=999999

```
>>>
3.14152314152
>>>
```

Whereas for $p=100$

```
>>>
```

```
3.24
```

```
>>>
```

Game of Life

The Game of Life, also known simply as Life, is a cellular automaton devised by the British mathematician John Horton Conway in 1970. The "game" is a zero-player game, meaning that its evolution is determined by its initial state, requiring no further input. One interacts with the Game of Life by creating an initial configuration and observing how it evolves.

Game of Life Rules

The universe of the Game of Life is an infinite two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, alive or dead. Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

First

Any live cell with fewer than two live neighbours dies, as if caused by under-population.

Second

Any live cell with two or three live neighbours lives on to the next generation.

Third

Any live cell with more than three live neighbours dies, as if by over-population.

Fourth

Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

Here is the code we wrote in class:

```
[frame=single]
```

```
import random
from graphics import *
```

#this function creates an NxN array filled with zeros

```
def empty(N):
    a=[]
    for i in range(N):
        b=[]
        for j in range(N):
            b=b+[0]
        a=a+[b]
    return a
```

#this function fills the array a with a portion p of live cells

```
def fill(a,p):
    N=len(a)
    for i in range(N):
        for j in range(N):
            if random.uniform(0,1)<p:
                a[i][j]=1
```

```
def update(A,B):
```

```
    N=len(A)
    for i in range(N):
        for j in range(N):
            neigh=A[(i-1)%N][(j-1)%N]+A[(i-1)%N][j]+A[(i-1)%N][(j+1)%N]+A[i][(j-1)%N]+A[i][(j+1)%N]+
            A[(i+1)%N][(j-1)%N]+A[(i+1)%N][j]+A[(i+1)%N][(j+1)%N]
            if A[i][j]==0:
                if neigh==3:
                    B[i][j]=1
                else:
                    B[i][j]=0
            else:
                if neigh==2 or neigh==3:
                    B[i][j]=1
                else:
                    B[i][j]=0
```

```
def gen2Dgraphic(N):
```

```
    a=[]
    for i in range(N):
        b=[]
        for j in range(N):
            b=b+[Circle(Point(i,j),.49)]
        a=a+[b]
```

```

    return a

def push(B,A):
    N=len(A)
    for i in range(N):
        for j in range(N):
            A[i][j]=B[i][j]

def drawArray(A,a,window):
    #A is the array of 0,1 values representing the state of the game
    #a is an array of Circle objects
    #window is the GraphWin in which we will draw the circles
    N=len(A)
    for i in range(N):
        for j in range(N):
            if A[i][j]==1:
                a[i][j].undraw()
                a[i][j].draw(window)
            if A[i][j]==0:
                a[i][j].undraw()

N=10          #N is the number of live cells you start with
win = GraphWin("title",400,400)
win.setCoords(-1,-1,N+1,N+1)
grid=empty(N)
grid2=empty(N)
circles=gen2Dgraphic(N)
fill(grid,0.3)

while True:
    drawArray(grid,circles,win)
    update(grid,grid2)
    push(grid2,grid)

```

simulations

The "game" is a zero-player game, meaning that its evolution is determined by its initial state, requiring no further input. One interacts with the Game of Life by creating an initial configuration and observing how it evolves or, for advanced players, by creating patterns with particular properties.

The universe of the Game of Life is an infinite two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, alive or dead. Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

Any live cell with fewer than two live neighbours dies, as if caused by under-population. Any live cell with two or three live neighbours lives on to the next generation. Any live cell with more than three live neighbours dies, as if by over-population. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

The initial pattern constitutes the seed of the system. The first generation is created by applying the above rules simultaneously to every cell in the seed—births and deaths occur simultaneously, and the discrete moment at which this happens is sometimes called a tick (in other words, each generation is a pure function of the preceding one). The rules continue to be applied repeatedly to create further generations.

python code

```
import random
from graphics import *

#this function creates an NxN array filled with zeros
def empty(N):
    a=[]
    for i in range(N):
        b=[]
```

QQ&ZιçL'Ė20151113134117.png

Figure 10: This is a type of game of life.

Any live cell with fewer than two live neighbours dies, as if caused by under-population.

Any live cell with two or three live neighbours lives on to the next generation.

Any live cell with more than three live neighbours dies, as if by over-population.

Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.


```

for j in range(N):
    b=b+[0]
    a=a+[b]
    return a

#this function fills the array a with a portion p of live cells
def fill(a,p):
    N=len(a)
    for i in range(N):
        for j in range(N):
            if random.uniform(0,1)<p:
                a[i][j]=1

def update(A,B):
    N=len(A)
    for i in range(N):
        for j in range(N):
            neigh=A[(i-1)%N][(j-1)%N]+A[(i-1)%N][j]+A[(i-1)%N][(j+1)%N]
            +A[i][(j-1)%N]+A[i][(j+1)%N]+A[(i+1)%N][(j-1)%N]+A[(i+1)%N]
            [j]+A[(i+1)%N][(j+1)%N]
            if A[i][j]==0:
                if neigh==3:
                    B[i][j]=1
                else:
                    B[i][j]=0
            else:
                if neigh==2 or neigh==3:
                    B[i][j]=1
                else:
                    B[i][j]=0

def gen2Dgraphic(N):
    a=[]
    for i in range(N):
        b=[]
        for j in range(N):
            b=b+[Circle(Point(i,j),.49)]
        a=a+[b]
    return a

def push(B,A):

```

```

N=len(A)
for i in range(N):
    for j in range(N):
        A[i][j]=B[i][j]

def drawArray(A,a,window):
    #A is the array of 0,1 values representing the state of the game
    #a is an array of Circle objects
    #window is the GraphWin in which we will draw the circles
    N=len(A)
    for i in range(N):
        for j in range(N):
            if A[i][j]==1:
                a[i][j].undraw()
                a[i][j].draw(window)
            if A[i][j]==0:
                a[i][j].undraw()

N=50
win = GraphWin("Title",600,600)
win.setCoords(-1,-1,N+1,N+1)
grid=empty(N)
grid2=empty(N)
circles=gen2Dgraphic(N)
fill(grid,0.1)

while True:
    drawArray(grid,circles,win)
    update(grid,grid2)
    push(grid2,grid)

```

The earliest interesting patterns in the Game of Life were discovered without the use of computers. The simplest static patterns and repeating patterns were discovered while tracking the fates of various small starting configurations using graph paper, blackboards, physical game boards and the like. During this early research, Conway discovered that the R-pentomino failed to stabilize in a small number of generations. In fact, it takes 1103 generations to stabilize, by which time it has a population of 116 and has fired six escaping gliders

Many different types of patterns occur in the Game of Life, in-

to open the window and draw 100x100 grid in it

cluding still lifes, oscillators, and patterns that translate themselves across the board. Some frequently occurring examples of these three classes are shown below, with live cells shown in black, and dead cells shown in white. Spaceships Glider Game of life animated glider.gif Lightweight spaceship Game of life animated LWSS.gif The "pulsar" is the most common period 3 oscillator. The great majority of naturally occurring oscillators are period 2, like the blinker and the toad, but oscillators of many periods are known to exist, and oscillators of periods 4, 8, 14, 15, 30 and a few others have been seen to arise from random initial conditions. Patterns called "Methuselahs" can evolve for long periods before stabilizing, the first-discovered of which was the R-pentomino. "Diehard" is a pattern that eventually disappears (rather than merely stabilizing) after 130 generations, which is conjectured to be maximal for patterns with seven or fewer cells. "Acorn" takes 5206 generations to generate 633 cells including 13 escaped gliders. Acorn Conway originally conjectured that no pattern can grow indefinitely—i.e., that for any initial configuration with a finite number of living cells, the population cannot grow beyond some finite upper limit. In the game's original appearance in "Mathematical Games", Conway offered a \$50 prize to the first person who could prove or disprove the conjecture before the end of 1970. The prize was won in November of the same year by a team from the Massachusetts Institute of Technology, led by Bill Gosper; the "Gosper glider gun" produces its first glider on the 15th generation, and another glider every 30th generation from then on. For many years this glider gun was the smallest one known. In 2015 a period-120 gun was discovered that has fewer live cells but a larger bounding box

Gosper glider gun Smaller patterns were later found that also exhibit infinite growth. All three of the following patterns grow indefinitely: the first two create one "block-laying switch engine" each, while the third creates two. The first has only 10 live cells. The second fits in a 5×5 square. The third is only one cell high:

Game of life infinite1.svg Game of life infinite2.svg

Game of life infinite3.svg Later discoveries included other "guns", which are stationary, and which shoot out gliders or other spaceships; "puffers", which move along leaving behind a trail of debris; and "rakes", which move and emit spaceships Gosper also constructed the first pattern with an asymptotically optimal quadratic growth rate, called a "breeder", or "lobster", which worked by leaving behind a trail of guns.

It is possible for gliders to interact with other objects in interesting ways. For example, if two gliders are shot at a block in just the right way, the block will move closer to the source of the gliders. If

three gliders are shot in just the right way, the block will move farther away. This "sliding block memory" can be used to simulate a counter. It is possible to construct logic gates such as AND, OR and NOT using gliders. It is possible to build a pattern that acts like a finite state machine connected to two counters. This has the same computational power as a universal Turing machine, so the Game of Life is theoretically as powerful as any computer with unlimited memory and no time constraints: it is Turing complete.

Furthermore, a pattern can contain a collection of guns that fire gliders in such a way as to construct new objects, including copies of the original pattern. A "universal constructor" can be built which contains a Turing complete computer, and which can build many types of complex objects, including more copies of itself