

# Introduction to Programming

Dr. Robert Kofler

March 4, 2015

# WHY PROGRAMMING?

- ▶ Biology is entering the era of "big-data"
  - ▶ Next generation sequencing (Genomes will soon be available for everyone, so what is my DNA telling me?)
  - ▶ Genomics, Transcriptomics, Proteomics, "-omics"
  - ▶ High throughput neural imaging (Connectomics)
  - ▶ Ecology
- ▶ You want to take back control over your computer? Who know's the frustrating feeling that the computer is doing what it want's and not what you want?
- ▶ Programming skills are a major advantage when looking for a job!
- ▶ Some do it for fun (you may even impress your girlfriend/boyfriend)

# QUICK JOB SEARCH

I did a quick job search for the term "Biomedicine" and I just displayed the 7 highest paying jobs (>120.000 Dollar per year)

## [Principal Cheminformatics Analyst \(JK13-01\)](#)

Forma Therapeutics - East Watertown, MA 02472

Experience with large volume , data centric processes involving scientific applications for storage, retrieval and analysis....

Monster - 6 days ago - [save job](#) - [email](#) - [more...](#)

## [Senior Scientist, Computer Aided Drug Design \(MT13-02\)](#)

Forma Therapeutics - Watertown, MA 02472

Excellent written and oral communication skills are required, as is the desire and ability to work in a multidisciplinary, hands-on, environment as an integral...

Monster - 18 days ago - [save job](#) - [email](#) - [more...](#)

## [Senior Scientist, Biophysical Sciences \(JE13-09\)](#)

Forma Therapeutics - East Watertown, MA 02472

The candidate must have the desire and ability to work in a multi-site, research-focused environment as an integral leader within the overall drug discovery...

Monster - 18 days ago - [save job](#) - [email](#) - [more...](#)

## [Senior Java/JEE Software Engineer](#)

SemanticBits - Herndon, VA 20171

Job Description SemanticBits is looking to hire a talented software developer who can help us build the next generation genetic testing platform This platform...

Dice - 16 days ago - [save job](#) - [email](#) - [more...](#)

# DO I REALLY NEED TO LEARN PROGRAMMING?



If you are happy with the above situation when faced with everyday biomedical data, no than not..

# WITH BIOINFORMATICS SKILLS: DATA ARTIST



# MINITASK

- ▶ Think for 10 minutes, about things you always wanted to do on the computer, but you could not!
- ▶ Write down everything you can think of
- ▶ Separate ideas into the categories university, job, private; All categories are equally welcome!! Just as an example I learnt programming because I needed to manage my mp3 collection...
- ▶ The content of the lecture is still flexible; Maybe we can specifically address some of your wishes

# WHY PYTHON?

- ▶ Easy to learn (relatively..)
- ▶ Very powerful! Can be used for small scripts and large software solutions!
- ▶ Many libraries are available: BioPython, PySam, NumPy, SciPy etc
- ▶ Runs on most operating systems: Windows, Mac, Linux
- ▶ Many great places are using it: Google, Lucasfilm (ILM), YouTube, NASA, **Institute of Populationgenetics**, half the bioinformaticians around the world
- ▶ it should be fun (the name Python is actually derived from Monty Python and not the snake)

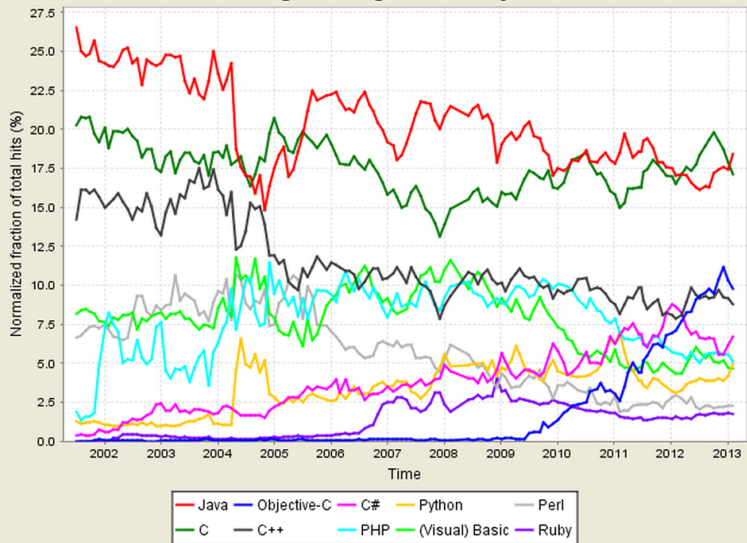
# SHORT HISTORY OF PYTHON

- ▶ Developed 1989 by Guido van Rossum in the Netherlands as successor to the ABC language (teaching language)
- ▶ Python 2.0 released in 2000
- ▶ Python 3.0 released in 2008; backwards incompatible = code from Python2 does not work with the Python3 interpreter; both versions are still in use!
- ▶ Python was the TIOBE programming language of the year in 2007 and 2010; This is awarded to the fastest growing language



# TIOBE INDEX

## TIOBE Programming Community Index



# AN ANNOYING ISSUE: PYTHON2 VS PYTHON3

## Considerations:

Most current code is in Python2.

Many libraries are for Python2 but they are (slowly) adapted to Python3

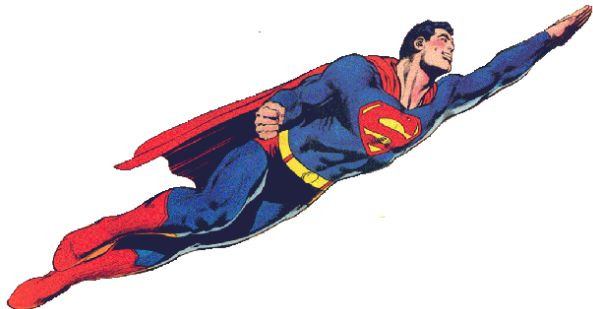
## We will learn Python3!

I do not want you to learn something which is outdated when you finish your studies.

Differences between Python2 and Python3 can be found here:

<http://docs.python.org/3/whatsnew/3.0.html>

# WHAT CAN WE EXPECT?



Is this realistic progress for one programming course?

# WHAT I HOWEVER DO EXPECT



That you make your first independent steps.. without help 😊

# MODE OF EXAMINATION

- ▶ During this lecture I will give you some tasks/homework. You will have some time to finish these tasks during the lecture.
- ▶ You collect all your tasks and bring them to the oral exam at the end of the year (USB, printed, or you can send me a zip)
- ▶ I will not give you solutions to these tasks for several reasons. First, for me it is more important that you try to do it on your own rather than to do it correctly. If you ever end up developing your own tools, you can not ask anyone.
- ▶ During the oral exam, I will ask you questions about your code. Especially, I will ask what will happen if I delete this line or if I move this part of the code to another position. Basically I want to know if you finished the tasks yourself.
- ▶ If you wrote all your scripts yourself you will certainly pass

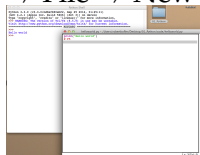
# AVAILABILITY OF LECTURE

<http://drrobertkofler.wikispaces.com/PythonLecture>



# LETS GET STARTED

Create a folder on the Desktop: 'python\_lecture'  
⇒ Finder ⇒ Applications ⇒ Python 3.3 ⇒ IDLE  
⇒ File ⇒ New Window



```
1 print("Hello world")  
2 # F5
```

⇒ File ⇒ Save as... ⇒ python\_lecture/helloworld.py  
⇒ Press F5



# MY FIRST SCRIPT

```
1 print("Hello, from my first script")
2 # comments start with '#'
3 nstr = input("Please give me a number ")
4 # the user has to provide a number
5 ni = int(nstr)
6 # the number has to be converted into a integer
7
8 print("Your number was",ni)
9 # lets print the given number
10 nmul = ni*ni # square the number
11 print("Your number squared is",nmul)
12 # lets print the square of the given number
13 # press F5 to execute the script
```

# CONTROL FLOW

How is the Python script executed? Many believe somehow magically all at once. But it is executed line-by-line!

```
print("Hello, from my first script")
```

```
nstr = input("Please give me a number ")
```

```
ni = int(nstr)
```

```
print("Your number was",ni)
```

```
nmul = ni*ni # square the number  
print("Your number squared is",nmul)
```



# METHODS AND RETURN VALUES

```
1 print("Welcome")
2 # print is a method
3 rstr = input("Guess the number: ")
4 # input is a method, which has a return value
5 # the return value is stored in the variable rstr
6
7 # variables can have any name e.g.:
8 # fritzi, hansi, ldkfhjaldkfj, t, t1, obladioblada
9 # exception 1: they must not start with a number
10 # exception 2: they must not be a python keyword
11 # eg: if, else, str, int, float, etc (colored in
    IDLE)
```

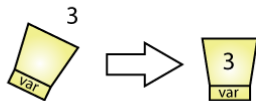
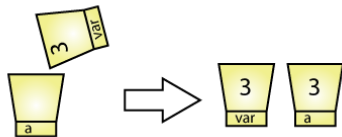
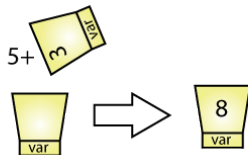
# CASTING

```
1 # another method
2 ni=int(nstr)
3
4 # the userinput g is a string (character sequence)
5 # needs to be converted to an integer (number)
6 # this conversion is called CASTING
```

# VARIABLE ASSIGNMENT

CODE

TRANSLATION

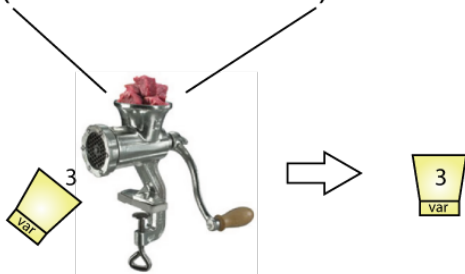
`var=3``a=var``var=var+5`

# VARIABLE ASSIGNMENT?

```
1 var=1
2 var=var+2
3 var=var+3
4 var=var+4
5 var=var+5
6 print(var)
7 # What will be printed here?
```

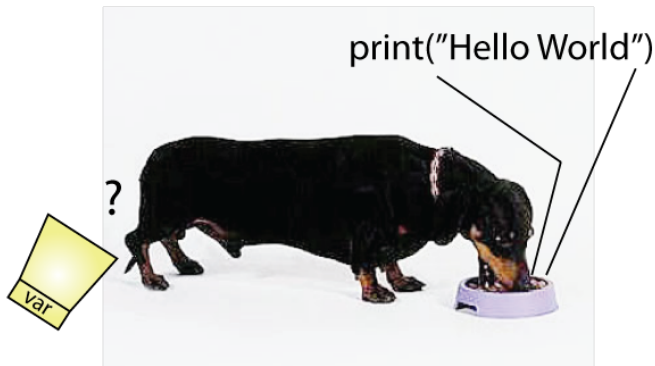
# FUNCTION WITH INTERESTING RETURN VALUE

```
var = input("Give me a number")
```



A function/method is a black box. Everything within brackets will be sent to the black box. The black box is doing some magic (processing the data) and sending the results back: "return value". In the above case "var" is collecting the return value

# DO I REALLY NEED TO COLLECT ALL RETURN VALUES?





# BASIC DATA TYPES

```
1 # String
2 s="hello world"
3
4 # Integer
5 i=2
6 # another Integer (variable names can have any length)
7 anseplbuaseimuada=3
8
9 # Float (floatting point number)
10 f=2.345
11
12 # Boolean
13 fridayrulez=True
14 weekendsucks=False
15
16 #let's print it
17 print(s,i,anseplbuaseimuada,f,fridayrulez,weekendsucks)
18 # save as basicdatatypes
19 # F5
```

# WHY IS IT IMPORTANT TO DISTINGUISH BETWEEN BASIC DATA TYPES

```
1 s="2"  
2 # what is the following doing?  
3 i=int(s)  
4  
5 stringsum=s+s  
6 intsum=i+i  
7  
8 print("Sum of strings:",stringsum)  
9 print("Sum of integers:",intsum)  
10  
11 # saves as importance_basicdata.py  
12 # F5  
13 # So what is going on here?
```

# BASIC OPERATIONS

- ▶ addition: +
- ▶ subtraction: -
- ▶ multiplication: \*
- ▶ division: /

# CASTING BETWEEN BASIC DATA TYPES

```
1 pi=3.1415927 # What's the name of this data type?
2 i=int(pi)
3 s=str(pi)
4 b=bool(pi) # What is going on here?
5
6 f=float(s)
7
8 print("Integer:", i)
9 print("String:", s)
10 print("Float:", f)
11 print("Boolean:", b)
12 # saves as morecasting.py
13 # Guess what will happen at each step!
14 # F5
```

Remember the methods, str, float, bool, int; Also remember that when casting to a boolean any value except zero is cast to True (only zero is False)

# TASK TURMRECHNEN

```
1 # For ANY user provided number the script should calculate:
2 """
3 Please give me a number: 5
4 You gave 5
5 *2= 10
6 *3= 30
7 *4= 120
8 *5= 600
9 *6= 3600
10 *7= 25200
11 *8= 201600
12 *9= 1814400
13 /2= 907200.0
14 /3= 302400.0
15 /4= 75600.0
16 /5= 15120.0
17 /6= 2520.0
18 /7= 360.0
19 /8= 45.0
20 /9= 5.0
21 This is what you gave no?
22 """
23 #Save as task-turmrechnen.py in your tasks folder!
```

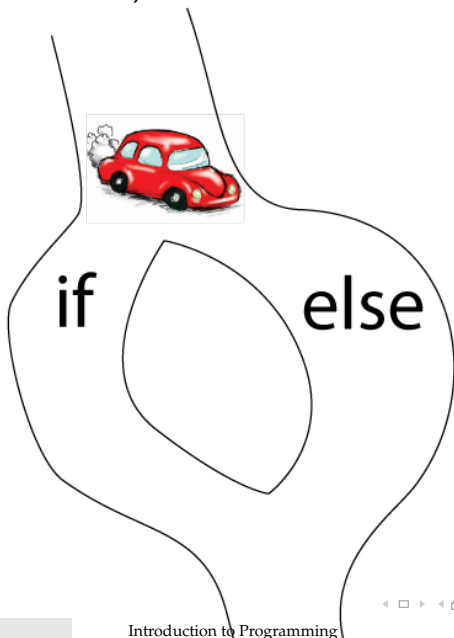
# A LITTLE GAME

```
1 print("Welcome")
2 g=input("Guess the number: ")
3 guess=int(g)
4
5 if guess==5:
6     print("You win!")
7 else:
8     print("You loose")
9 print("Game over")
10 # save as alittlegame.py
11 # F5
12 # so let's guess what is this code doing?
```

# CONTROL STRUCTURE: IF-ELSE

```
1 # check if value of variable guess
2 # equals 5
3 # if so print you win else you loose
4 if guess==5:
5     print("You win!")
6 else:
7     print("You loose")
```

# DECIDE: IF OR ELSE, NOT BOTH

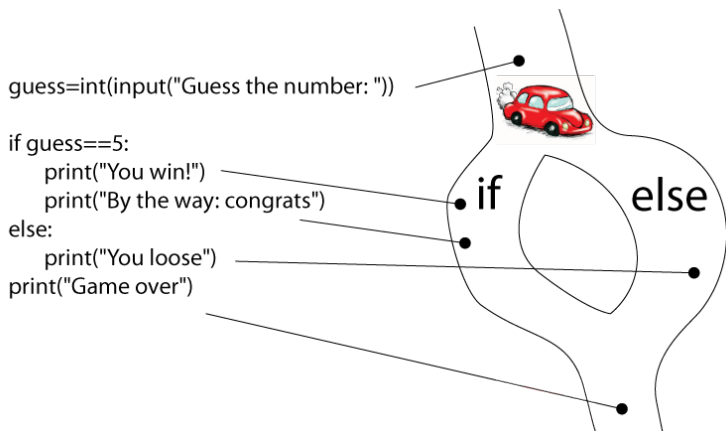




# MORE IF-ELSE

```
1 guess=int(input("Guess the number: "))
2
3 if guess==5:
4     print("You win!")
5     print("By the way: congrats")
6 else:
7     print("You loose")
8 print("Game over")
9 # Save as simpleifelse.py F5
```

# CONTROL FLOW VISUALIZED



# SO WHAT WILL HAPPEN NOW?

```
1 # What is happening now?
2 guess=int(input("Guess the number: "))
3
4 if guess==5:
5     print("You win!")
6     print("By the way: congrats")
7 else:
8     print("You loose")
9     print("Game over")
```

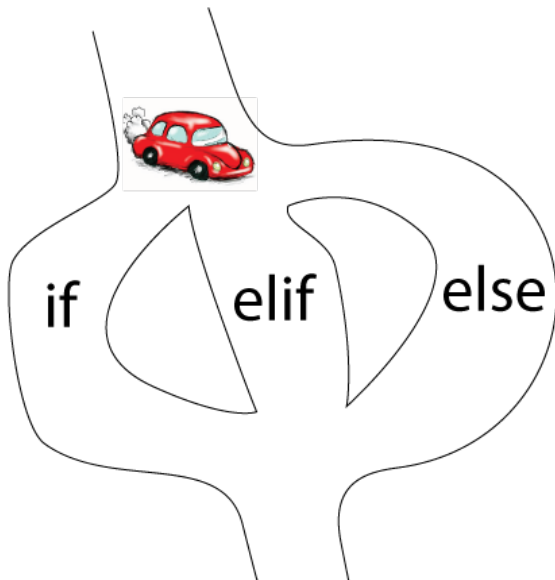
Remember: Intendation is everything!!

# THE MANY FACES OF IF-ELSE

- ▶ if
- ▶ if-else
- ▶ if-elif-else
- ▶ if-elif...elif-else

```
1 # example
2 if guess==1:
3     print("you guessed one")
4 elif guess==2:
5     print("you guessed two")
6 elif guess==3:
7     print("you guessed three")
8 else:
9     print("You guess nonsense")
```

# IF..ELIF..ELSE VISUALIZED



# A SMALL BUT BIG DIFFERENCE: = VS ==

```
1 guess = int(input("Guess a number: "))
2 guesscorrect = guess == 5
3 if guesscorrect:
4     print("You win!")
5 else:
6     print("You loose")
7 print("Guess:", guess)
8 print("Guesscorrect", guesscorrect)
9 # Save as smallbigdifference.py
10 # F5
```

# COMPARISON OPERATORS IN PYTHON

- ▶ equals: `==`
- ▶ not equals: `!=`
- ▶ larger than: `>`
- ▶ less than: `<`
- ▶ larger than or equal to: `>=`
- ▶ less than or equal to: `<=`

# SOMETHING IS WRONG: FIX THIS SCRIPT

```
1 # Hmm, something is wrong?
2 # Can you find the problem
3
4 number = int(input("Give me a number: "))
5 if number > 3:
6     print("Number larger than three")
7 elif number > 7:
8     print("Number larger than seven")
9 else:
10    print("Number smaller or equal to three")
11 print("Finished")
12 # save as ifelif.py
13 # F5
```



# STANDALONE IF

```
number=int(input("Give me a number"))  
print("You gave:",number)
```

```
if(number>5):  
    print("Which is larger than 5")
```

```
print("Done")
```



if

# ANOTHER TASK

```
1 # Task: There is something wrong, fix me
2 # Hint 1: I only want to have one output line
3 # Hint 2: The most accurate description of my number
4 number=int(input("Give me a number "))
5 intro="Your number is "
6 if number > 2:
7     print(intro+"larger than two")
8     if number > 4:
9         print(intro+"larger than four")
10        if number > 6:
11            print(intro+"larger than six")
12            if number > 8:
13                print (intro+"larger than eight")
14            else:
15                print(intro+"smaller than two")
16 # save as task-if.py in tasks folder
17 # F5
```

# QUESTIONS?

Any questions about the previous tasks?

# BACK TO OUR GAME

```
1 print("Welcome")
2 g=input("Guess the number: ")
3 guess=int(g)
4
5 if guess==5:
6     print("You win!")
7 else:
8     print("You loose")
9 print("Game over")
10 # save as alittlegame.py
11 # F5
12 # so let's guess what is this code doing?
```

So now, it's your turn, please explain the code to me!

This game is a bit boring, any ideas of how we could improve it?

# IMPROVED GAME

```
1 import random
2 # we are using the library random
3 print("Welcome")
4 guess=int(input("Guess the number: "))
5 randomnumber= random.randint(1,5)
6 # From the library random we are using the function
   randint
7 # Imagine the point as: "from random give me a
   subfunction"
8 if guess==randomnumber:
9     print("You win!")
10 else:
11     print("You loose, the number was",randomnumber)
12 print("Game over")
13 # save as improvedgame.py
14 # F5
```

# LIBRARIES

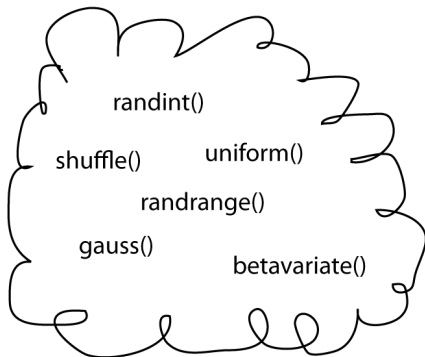
Python offers many standard libraries

- ▶ random: Generate random numbers
- ▶ math: Mathematics library, e.g.: square root
- ▶ sys: System access, e.g.: get command line parameter
- ▶ re: Regular expressions for easy text parsing
- ▶ collections: for storing data
- ▶ argparse: parsing command line parameters
- ▶ tkinter: graphical user interface
- ▶ doctest: test your code

# THE MEANING OF THE DOT "."

Random...what??

RANDOM:



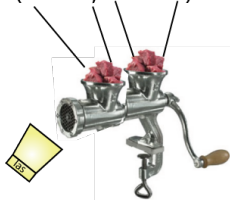
random.randint()

# METHOD WITH TWO PARAMETERS: RANDINT(1,5)?

```
var = input("Give me a number")
```



```
las = createlasagne("beef", "horse")
```



Careful with the order, the order of parameters matter. For example imagine that the 'createlasagne' adds 95% of the first type of meat, and 5% of the second type. Accidentally reversing the order would result in..???



# INFORMATION ABOUT PYTHON CORE LIBRARY

So if you do not know what a function is going to do, which parameters are need and in which order...than you find the solution at:

<http://docs.python.org/3/>

E.g.: the entry about `randint()`

```
random.randint(a, b)
```

Return a random integer  $N$  such that `a <= N <= b`. Alias for `randrange(a, b+1)`.

Functions for sequences:

# FINALLY PROGRAMMING PAYS OFF

We sold our game to a Las Vegas casino!



# A CALL FROM LAS VEGAS EMERGENCY!!!

Something is wrong, we are loosing money! We are going to sue you!



# WHO'S FAULT IS IT?

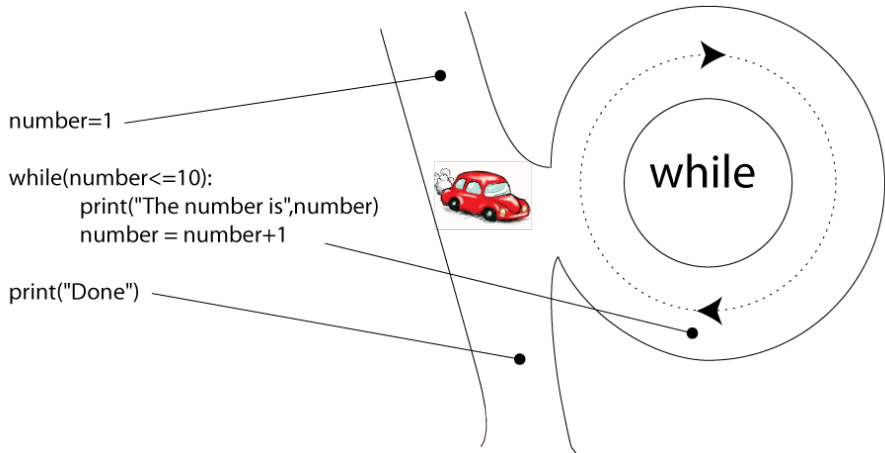
They blame it on the `randint()`. We need to prove that the `randint` is fine!



# BUT, MEAN'WHILE'

```
1 number=1
2
3 while(number<=10):
4     print("The number is",number)
5     # in the end we increment the number
6     number = number+1
7
8 print("Done")
9 # safe as whiledemo.py
10 # F5
```

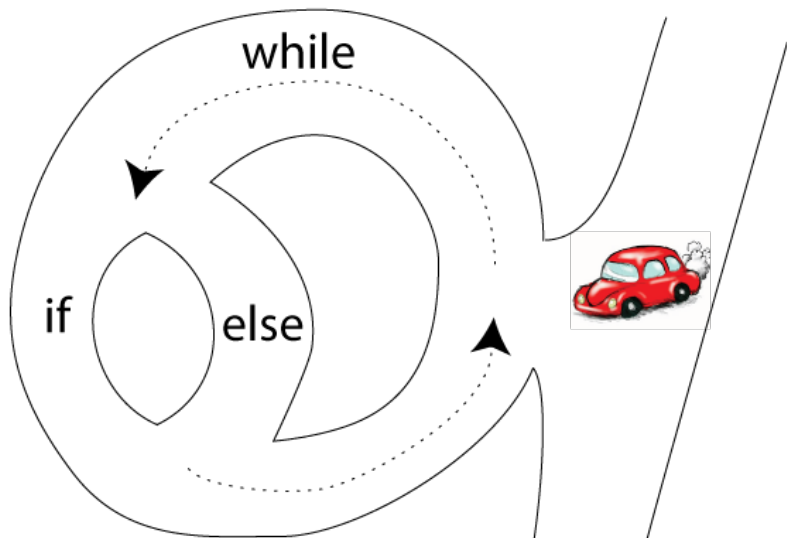
# TRAPPED IN THE ROUNDABOUT



# IF ELSE IN WHILE

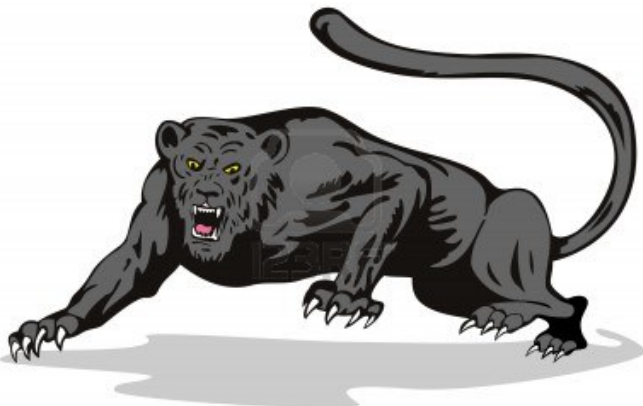
```
1 # %..the modulo operator
2 # it gives the remainder of a division
3 # eg: 10 % 3 = 1
4 number=1
5 while (number<=10) :
6     if (number%2==0) :
7         print (number, " is even")
8     else:
9         print (number, " is uneven")
10    number=number+1
11 print ("Done")
12 # safe as whilemodulo.py
13 # F5
```

## HMM...A ROUNDABOUT WITH A CROSSING??





# NOW, YOU ARE READY FOR PAYBACK



Demonstrate to the Las Vegas guy's that our random number generator works just fine!

# TASK4: PAYBACK

```
1 # Task: demonstrate that our random number generator
   works!
2 # Generate 1000 random numbers and count the occurrences
   of each
3 # the output should be similar to this:
4 """
5 Counts of 1: 208
6 Counts of 2: 193
7 Counts of 3: 185
8 Counts of 4: 211
9 Counts of 5: 203
10 Sum: 1000
11 """
12 # Hint: use while loop and if-elif
13 # safe as task-random5.py in your tasks folder
14 # F5
```

## CALMING DOWN

After demonstrating that our `randint()` works just fine the situation in Las Vegas has relaxed a bit.



# NEW IDEA FOR EARNING EVEN MORE MONEY



Las Vegas called again! If the random number generator would produce numbers between 1 and 100, they may introduce a larger profit margin. But they insist that we make absolutely sure that the random number generator works. We need to test `randint(1,100)`... Think about the solution for `randint(1,5)` – > PANIK!

# INTRODUCING TUPLE AND LIST

```
1 # two new data types
2
3 # tuple = immutable
4 a=(1,2,3,4,5)
5
6 # list = mutable
7 b=[6,7,8,9,10]
8
9 print(a)
10 print(b)
11
12 # save as tuplelist.py
13 # F5
```

# I FORGOT, WHAT DATA TYPE WAS IT AGAIN?

```
1 # introducing the type() function
2 a=(1,2,3,4,5)
3 b=[6,7,8,9,10]
4
5 print(a)
6 print(type(a))
7 print(b)
8 print(type(b))
9
10 # save as temp.py
11 # F5
```

# ACCESSING THE ELEMENTS OF A TUPLE/LIST

```
1 a=(1,2,3,4,5)
2 b=[6,7,8,9,10]
3 azero=a[0]
4 bone=b[1]
5
6 print(a)
7 print("a-zero: ",azero)
8
9 print(b)
10 print("b-one: ",bone)
11 # save as access.py
12 # Noticed anything strange?
13 # F5
```

# FORGET WHAT YOU LEARNED IN KINDERGARTEN

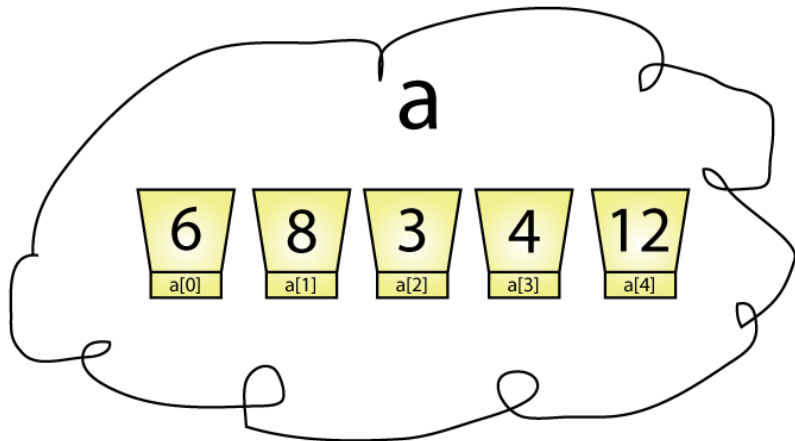
In kindergarten you learnt to count: 1, 2, 3, 4, 5,...

**JUST FORGET IT!!**

In programming it's **WRONG**, it has to be: 0, 1, 2, 3, 4, 5,...



## ILLUSTRATING TUPLE/LIST

 $a = (6, 8, 3, 4, 12)$ 

# CHANGING LIST CONTENT

```
1 a=[1,2,3,4,5]
2 print(a)
3
4 a[0]=12 # what will happen?
5 print(a)
6
7 a[4]=a[0] # what will happen?
8 print(a)
9
10 a[0]=a[0]+1 # what will happen?
11 print(a)
12
13 # save as changinlist.py
14 # F5
```

# CHANGING TUPLE CONTENT?

```
1 a=(1,2,3,4,5)
2 print(a)
3
4 a[0]=12 # what will happen?
5 print(a)
6
7 a[4]=a[0] # what will happen?
8 print(a)
9
10 a[0]=a[0]+1 # what will happen?
11 print(a)
12 # F5
```

# INTRODUCING: FOR

```
1 # create a list
2 somelist=[10,3,5,10,50,63,40]
3 print(somelist)
4
5 # iterate over every element in a list
6 for element in somelist:
7     print("The next element is",element)
8 # save as forloop.py
9 # F5
```

## RANGE: CREATING (KIND OF) A LIST

```
1 # the range method
2 somerange=range(0,10)
3 somelist=list(somerange) # convert range to
   list
4
5 print(somerange)
6 print(somelist)
7
8 # iterate over range-object
9 for element in somerange:
10     print("Element of range",element)
11 # iterate over list
12 for element in somelist:
13     print("Element of list",element)
14 # save as range.py
15 # F5
```

# SUMMARY: CONTROL STRUCTURES

- ▶ `if..elif..elif..else`: create alternative paths (roads) in your program
- ▶ `while(booleanstatement)`: repeat something as long as the boolean statement is true
- ▶ `for something in somecontainer`: iterate over all elements in some container

That's basically it! You only need these three control structures to write a arbitrary complex program.

# ONE LAST THING: CREATING DEFAULT LISTS/TUPLES

```
1 # creating lists of a given size
2 a=[0,] * 5
3 print(a)
4
5 # creating a tuple of a given size
6 b = (1,) * 20
7 print(b)
8
9 c = (1) * 100 # Guess what will happen?
10 print(c)
11 # What is going on here?
12 # Any explanations?
13 # F5
```

# READ FOR THE EXTRA DOLLARS?

Now you know everyting required for testing randint(1,100).





## TASK 5: TESTING RANDINT(1,100)

```
1 # Task: Test randint(1,100)
2 # Create 10.000 random numbers and count the occurrence of each
3 # Than create an output similar to the following:
4 """
5 Count of 1 is 110
6 Count of 2 is 101
7 Count of 3 is 99
8 Count of 4 is 104
9 Count of 5 is 95
10 Count of 6 is 112
11 Count of 7 is 99
12 Count of 8 is 102
13 ....etc....
14 """
15 # Hint1: start with a default-list..
16 # Hint2: use 'for..in..' to print the values of the list
17 # Save as task-randint100.py in your tasks folder
18 # F5
```

# IMPROVING THE GAME

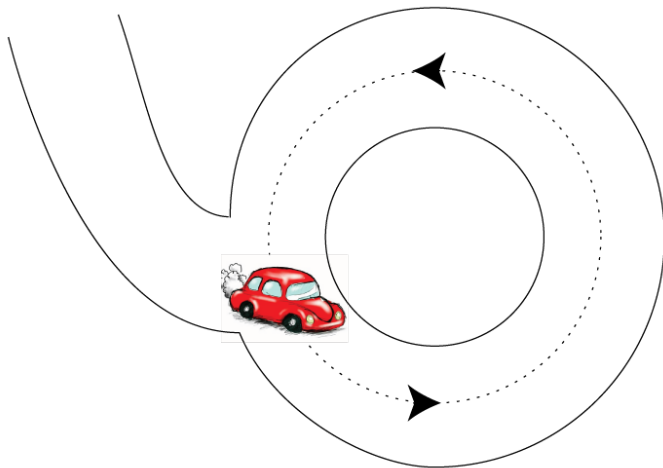
```
1 import random
2
3 print("Welcome")
4 guess=int(input("Guess the number: "))
5 randomnumber = random.randint(1,100)
6 if guess==randomnumber:
7     print("You win!")
8 else:
9     print("You loose, the number was",randomnumber)
10 print("Game over")
11 # save as improvedgame.py
12 # F5
```

Nice, but a bit hard to win. How could we improve this game?

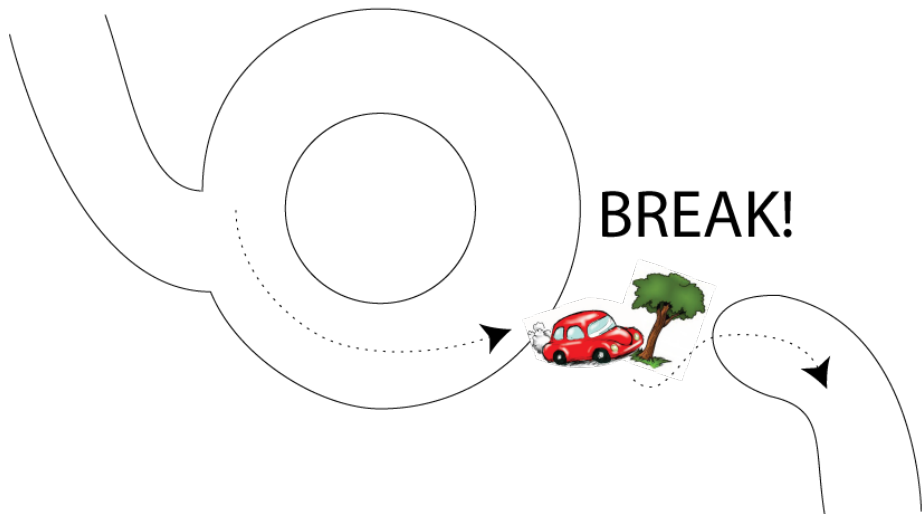
# FURTHER IMPROVEMENTS

```
1 import random
2 print("Welcome")
3 randomnumber = random.randint(1,100)
4 attempts=1
5 while(True):
6     guess=int(input("Guess the number: "))
7     if guess==randomnumber:
8         print("You win, after",attempts,"attempts")
9         break
10    elif guess < randomnumber:
11        print("The number is larger")
12    else:
13        print("The number is smaller")
14    attempts+=1
15    # lacy man's syntax for: attempts=attempts+1
16 print("Game over")
17 # save as improvedgame.py
18 # F5
```

# TRAPPED IN WHILE(TRUE)



# POSSIBLE ESCAPE ROUTES?



# PLAY AGAIN?

```
1 import random
2 print("Welcome")
3 while(True):
4     randomnumber = random.randint(1,100)
5     attempts=1
6     while(True):
7         guess=int(input("Guess the number: "))
8         if guess==randomnumber:
9             print("You win, after",attempts,"attempts")
10            break
11        elif guess < randomnumber:
12            print("The number is larger")
13        else:
14            print("The number is smaller")
15        attempts+=1
16    playagain=input("Wanna play again? y/n")
17    if(playagain=="n"):
18        break
19    else:
20        print("Great a new game..")
21 print("Game over")
22 # save as finalgame.py
23 # F5
```

# GAMING LIKE A PRO



Starting a Python script from the command line!

# OPEN THE COMMAND LINE

⇒ Finder ⇒ Applications ⇒ Utilities ⇒ Terminal

```
Terminal - zsh - 105x22
-rw-r--r--@ 1 robertkofler staff 242104533 Apr 30 11:23 s0025-N2000.predictions
-rw-r--r--@ 1 robertkofler staff 40239820 Apr 30 11:03 s0025-N250.labels
-rw-r--r--@ 1 robertkofler staff 178097834 Apr 30 11:21 s0025-N250.predictions
-rw-r--r--@ 1 robertkofler staff 58157040 Apr 30 11:05 s0025-N4000.labels
-rw-r--r--@ 1 robertkofler staff 262941324 Apr 30 11:34 s0025-N4000.predictions
-rw-r--r--@ 1 robertkofler staff 44762340 Apr 30 11:04 s0025-N500.labels
-rw-r--r--@ 1 robertkofler staff 201875225 Apr 30 11:22 s0025-N500.predictions
-rw-r--r--@ 1 robertkofler staff 62581140 Apr 30 11:05 s0025-N8000.labels
-rw-r--r--@ 1 robertkofler staff 283318879 Apr 30 11:34 s0025-N8000.predictions
-rw-r--r--@ 1 robertkofler staff 49197620 Apr 30 11:02 s01-N1000.labels
-rw-r--r--@ 1 robertkofler staff 220673831 Apr 30 11:12 s01-N1000.predictions
-rw-r--r--@ 1 robertkofler staff 53677400 Apr 30 11:02 s01-N2000.labels
-rw-r--r--@ 1 robertkofler staff 239785192 Apr 30 11:13 s01-N2000.predictions
-rw-r--r--@ 1 robertkofler staff 40239820 Apr 30 10:49 s01-N250.labels
-rw-r--r--@ 1 robertkofler staff 176876876 Apr 30 11:10 s01-N250.predictions
-rw-r--r--@ 1 robertkofler staff 58157040 Apr 30 11:02 s01-N4000.labels
-rw-r--r--@ 1 robertkofler staff 260094904 Apr 30 11:13 s01-N4000.predictions
-rw-r--r--@ 1 robertkofler staff 44762340 Apr 30 11:02 s01-N500.labels
-rw-r--r--@ 1 robertkofler staff 200758538 Apr 30 11:11 s01-N500.predictions
-rw-r--r--@ 1 robertkofler staff 62581140 Apr 30 11:03 s01-N8000.labels
-rw-r--r--@ 1 robertkofler staff 280264699 Apr 30 11:14 s01-N8000.predictions
[0,11310]robertkofler% /Volumes/Temp1/simcoal2/papsim-Ne/simres
```

The color can be adjusted and is simply a matter of taste. Per default it is black!



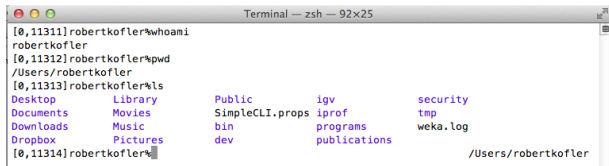
# THE THREE MAJOR QUESTIONS



- ▶ Who am I?
- ▶ Where am I?
- ▶ Who is the strange person in my bed?

# YOUR FIRST UNIX COMMANDS

- ▶ 'whoami' ⇒ Who am I?
- ▶ 'pwd': print working directory; which folder is my current position; ⇒ Where am I?
- ▶ 'ls': list the files in my current directory ⇒ Who is the strange person in my bed?



```
Terminal — zsh — 92x25
[0,11311]robertkofler%whoami
robertkofler
[0,11312]robertkofler%pwd
/Users/robertkofler
[0,11313]robertkofler%ls
Desktop      Library      Public      igv          security
Documents    Movies       SimpleCLI.props  iprof        tmp
Downloads    Music        bin          programs     weka.log
Dropbox      Pictures     dev          publications
[0,11314]robertkofler%  
/Users/robertkofler
```

# STUDYING THE STRANGE PERSON IN MORE DETAIL

`ls -l` ⇒ list long (-l)

You are thus providing a parameter to the list command.

How did you provide a parameter to Python methods?

```
Terminal — zsh — 102x15
drwxr-xr-x  9 robertkofler  staff   306 Jan 21 12:06 syn-nonsyn
[0,11322]robertkofler% ls -l
total 240
drwxr-xr-x  17 robertkofler  staff   578 Apr 30 14:41 Modules
-rw-r--r--   1 robertkofler  staff  4576 Jan 21 12:06 VarSliding2Flybase.pl
-rw-r--r--   1 robertkofler  staff  5761 Jan 21 12:06 VarSliding2Wiggle.pl
-rw-r--r--   1 robertkofler  staff 21960 Jan 21 12:06 Variance-at-position.pl
-rwxr-xr-x   1 robertkofler  staff  8279 Jan 21 12:06 Variance-for-feature.pl
-rw-r--r--   1 robertkofler  staff 25265 Jan 21 12:06 Variance-sliding.pl
-rw-r--r--   1 robertkofler  staff  5710 Jan 21 12:06 Visualize-output.pl
drwxr-xr-x   9 robertkofler  staff   306 Mar 20 11:25 basic-pipeline
-rw-r--r--   1 robertkofler  staff 19610 Jan 21 12:06 calculate-dxy.pl
-rw-r--r--   1 robertkofler  staff 13124 Jan 21 12:06 mauve-parser.pl
drwxr-xr-x   9 robertkofler  staff   306 Jan 21 12:06 syn-nonsyn
[0,11322]robertkofler% /Users/robertkofler/dev/popoolation
```

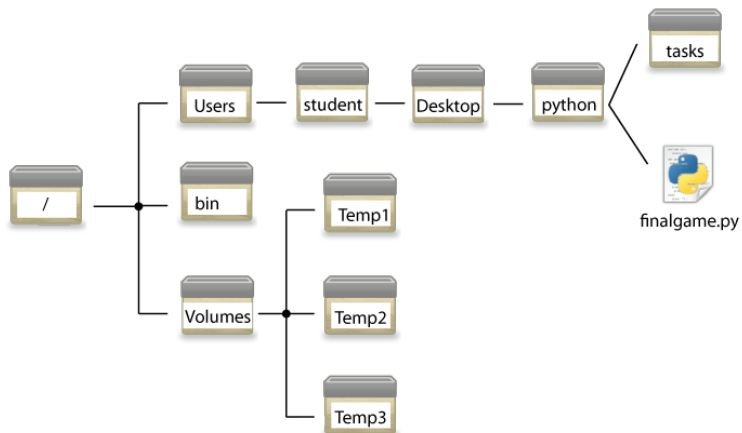
- ▶ last column: Name of the file/directory
- ▶ first column starts with a "d": directory
- ▶ first column ends with a x: directly executable file (Attention: Python scripts are not per default directly executable.)
- ▶ column 3: owner of the file/directory
- ▶ column 5: size of the file in bytes ("`ls -l -h`" or "`ls -lh`" ⇒ human readable)

# PLANNING YOUR ESCAPE (WALK OF SHAME)

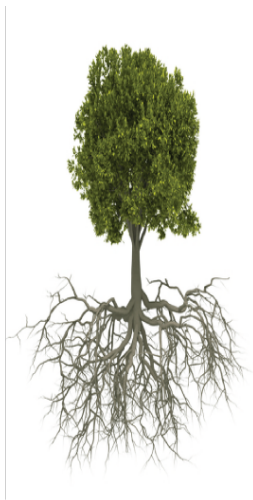
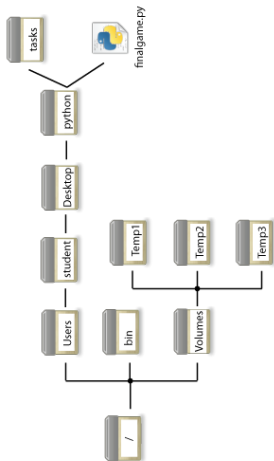
**cd** ⇒ change directory

- ▶ 'cd bed': move into folder bed; relative path
- ▶ 'cd ..': move to the previous folder; relative path
- ▶ 'cd /': move to the root folder ⇒ **THE GOD FOLDER**; absolute path
- ▶ 'cd /Users/student': move into the student folder; absolute path

## EXAMPLE TREE



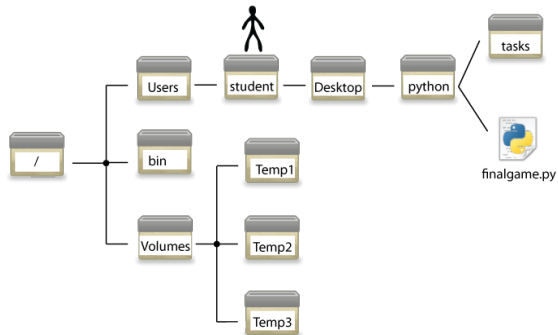
# WHY ROOT?



All the directories and files are sprouting from 'root'. However, trunk would probably be a better name..

# CHANGING THE FOLDER #1

We are at 'student' and want to go to 'Desktop'

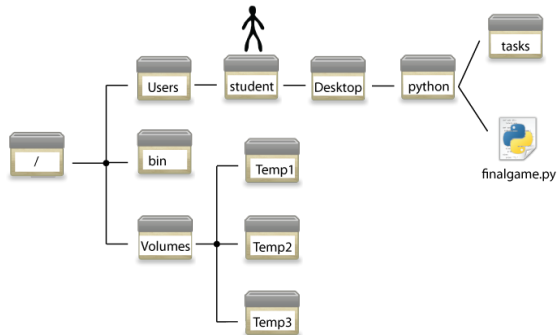


relative path: 'cd Desktop'

absolute path: 'cd /Users/student/Desktop'

## CHANGING THE FOLDER #2

We are at 'student' and want to go to our 'tasks' folder



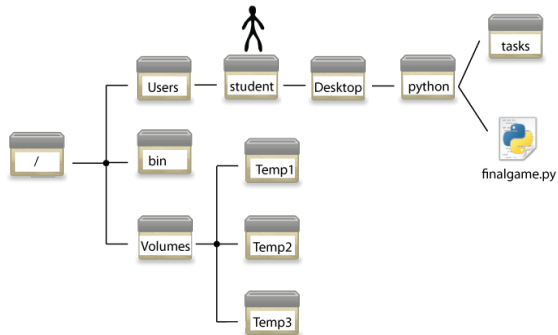
relative path: `'cd Desktop/python/tasks'`

absolute path: `'cd /Users/student/Desktop/python/tasks'`



## CHANGING THE FOLDER #3

We are at 'student' and want to go to the 'Users' folder

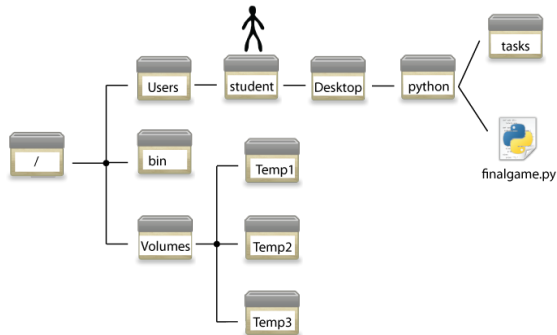


relative path: `'cd ..'`

absolute path: `'cd /Users'`

# CHANGING THE FOLDER #4

We are at 'student' and want to go to **THE FOLDER** (How is it called?)

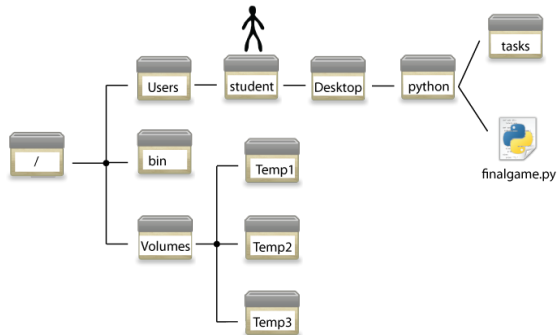


relative path: `'cd ../../'`

absolute path: `'cd /'`

# CHANGING THE FOLDER #5

We are at 'student' and want to go to the 'Temp1' folder

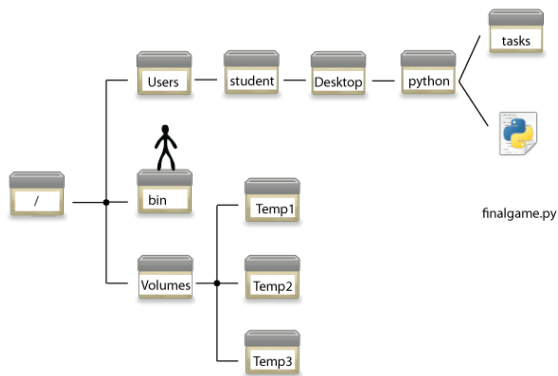


relative path: `'cd ../../Volumes/Temp1'`

absolute path: `'cd /Volumes/Temp1'`

# CHANGING THE FOLDER #6 - YOUR TURN

You are at 'bin' and want to go to the 'python' folder



relative path: ?  
absolute path: ?

## SUMMARY: ABSOLUTE PATH - RELATIVE PATH

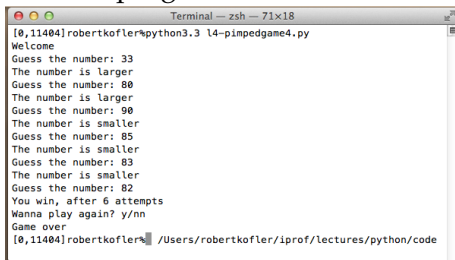
- ▶ : absolute path: always starts with root symbol '/' and is ALWAYS the same for any given directory, irrespective to your position in the directory tree. Is more conservative, mixing up files is more difficult. For my scripts I frequently provide absolute paths as parameters because they are unambiguous.
- ▶ : relative path: never starts with the root symbol. It is always relative to your position. May be more convenient because usually you have to type less.

## GAMEING LIKE A PRO

Now it's time to start your game like a real professional.

- ▶ First change to the folder with the script, it should be: 'cd Desktop/python\_lecture'
- ▶ Than start your script: 'python3.3 finalgame.py'; We are calling Python3 and simply passing our script as an argument!
- ▶ Finally enjoy your game

Congrat's you are now officially a nerd :) - btw, no point denying it, you just started your own program from the command line

A terminal window titled "Terminal — zsh — 71x18" showing the execution of a Python script. The script prompts the user to guess a number between 1 and 100. The user makes six guesses: 33, 80, 90, 85, 83, and 82. The script provides feedback: "The number is larger" for the first three guesses and "The number is smaller" for the last two. After the sixth guess, the script says "You win, after 6 attempts" and asks "Wanna play again? y/nn". The user enters "y", and the script says "Game over". The prompt returns to the user's shell.

```
Terminal — zsh — 71x18
[0,11404]robertkofler%python3.3 l4-pimpedgame4.py
Welcome
Guess the number: 33
The number is larger
Guess the number: 80
The number is larger
Guess the number: 90
The number is smaller
Guess the number: 85
The number is smaller
Guess the number: 83
The number is smaller
Guess the number: 82
You win, after 6 attempts
Wanna play again? y/nn
Game over
[0,11404]robertkofler% /Users/robertkofler/iprof/lectures/python/code
```

## NEXT TIME: GENE INSPECTOR

Next time it will get biomedical and we will start inspecting the human genes!

