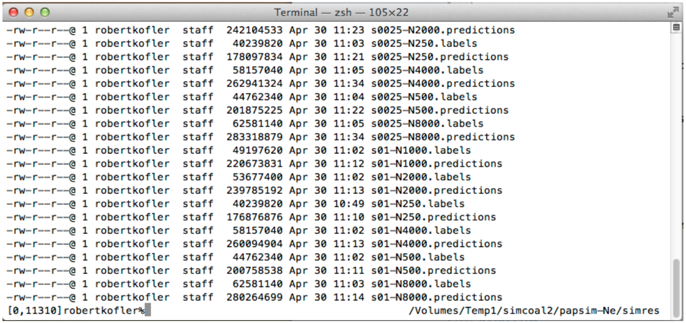
LAST TIME..THE COMMAND LINE

Maybe slightly to fast..
⇒ Finder ⇒ Applications ⇒ Utilities ⇒ Terminal



```
-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 final game

Store in python folder

```python
import random
print("Welcome")
while(True):
    randomnumber = random.randint(1,100)
    attempts=1
    while(True):
        guess=int(input("Guess the number: "))
        if guess==randomnumber:
            print("You win, after",attempts,"attempts")
            break
        elif guess < randomnumber:
            print("The number is larger")
        else:
            print("The number is smaller")
        attempts+=1
    playagain=input("Wanna play again? y/n")
    if(playagain=="n"):
        break
    else:
        print("Great a new game..")
print("Game over")
# save as finalgame.py
# F5
```

Reminder: Unix commands

- 'whoami'
- 'pwd': print working directory
- 'ls': list the files in my current directory
- 'cd': change directory
- 'python3.3': start a script using the Python3 interpreter

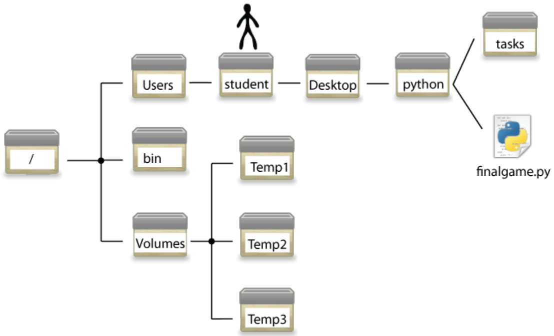
Start the game from the Unix-shell

↩ = Enter button

- cd Desktop/pythonlecture *(= change to the path where the script is stored)*
- in case your folder has a different name use ″ls -l″ and ″cd″ to navigate into the folder containing your python script
- python3.3 finalgame.py ↩ *(= tell python3 to interpret your script)*
- Enjoy your game

In case you see something weird like ″>>>″ you accidentially entered the Python command line. Type ″quit() ↩ ″ to escape

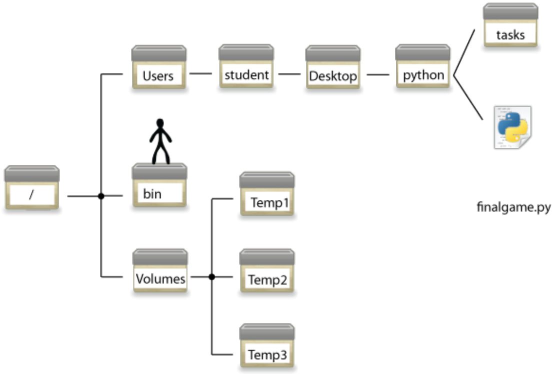
Start a python script directly from student



python3.3 Desktop/python/finalgame.py ↩

Starting a python script from anywhere using absolute paths

You are at 'bin' and want to start the game



```
python3.3 /Users/student/Desktop/python/finalgame.py ↩
```

Mini task: Start your python script using absolute paths

python3.3 /Users/student/Desktop/python/finalgame.py $\boxed{\hookleftarrow}$
Hint: use $\boxed{\leftrightarrows}$ (= Tabulator) for autocomplete!

- type: "python3.3 /U" $\boxed{\leftrightarrows}$
- continue: "python3.3 /Users/s" $\boxed{\leftrightarrows}$
- continue: "python3.3 /Users/student/D" $\boxed{\leftrightarrows}$
- ...

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.

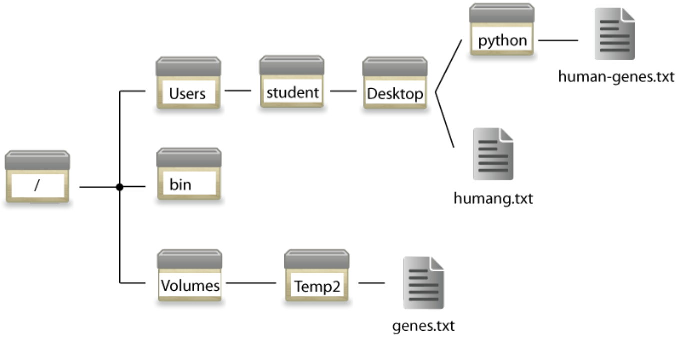
Played enough

LET'S GET BIOMEDICAL

DOWNLOADING THE HUMAN GENES

http://drrobertkofler.wikispaces.com/PythonLecture
$\Rightarrow$ store in your python folder

THE ABSOLUTE PATH OF FILES



genes.txt: /Volumes/Temp2/genes.txt
humang.txt: /Users/student/Desktop/humang.txt
human-genes.txt: ?

MINI TASK: FIND THE ABSOLUTE PATH OF YOUR HUMAN-GENES FILE

- First open a new IDLE window.
- Copy the absolute path as a comment to IDLE.
- Hint: use the command line, navigate into the folder containing your genes-file and type ’pwd’ to get the directory. You can than manually enter the rest.
- Hint: if you are lazy, use autocomplete

Printing the content of the genes file

```python
genepath="/Users/robertkofler/Desktop/python/human-genes.gtf"

# we use a for-loop to iterate over the file, line by line
for line in open(genepath,"r"):
    print(line)

# save as geneprinter.py -> F5
# press Ctrl - c to stop it
```

FILE OBJECT - ANOTHER OBJECT

If you want to read a file, you need the following information:

- position of the file = absolute path
- read the file or write to the file?
- what is my current position in the file, i.e.: which line have I last read.

All this information is stored in the file object! You do not have to worry about the current position in the file, you are just reading line by line.
The file object is thus another object, so which objects have we met so far?

WHY OBJECTS

Helps you modelling the real world. Imagine you are programming a farm-simulator (Farmville 3). You would create an object for a horse, a cow, a barn, the farmer, of course also for the farmer/innen, the pond, etc.. It thus allows you to develop the software at a very picturesque (tangible level). Instead of loops within loops, you have cows interacting with farmers (nice level ob abstraction).

WHY DO WE NEED OBJECTS?

Unfortunately there is no escape from objects in Python: file object, range even the string..
Later we will learn how to create objects, but now we will just make first contact.
And remember in addition to the basic data types int, float, list, tuple we now add infinitely many different types of objects..

FORMAL DEFINITION OF AN OBJECT

Objects in "object-oriented programming" are essentially data structures together with their associated processing routines.
A object thus has:

- data structures
- processing routines (=called methods; similar to functions)

Modeling a cow

Which data-structures (which information does a cow have)?

- age
- milk yield
- ?? any other ideas

Which formal routines (what can you do with a cow)?

- feed()
- milk()
- ??

CLASS VS INSTANCE

Class
All cows adhere to the same template; the all have a age, a milk yield and so on, this template is called ”class”.
Instance
One specific cow adhering to the template (class). Thus the cow named ”Resi” is an instance, a cow named ”Pepi” is an instance and so on.
To summarize, a class ’cow’ represents the general concept of a cow, an instance of a ’cow’ is one particular cow.

```
# Create an instance of a cow Cow()
resi = Cow(12, 4000) # age, milk yield
resi.feed(2) # give 2 kilo of grass
# note the point "."
# with the point we are calling a method for
# the particular instance
milk = resi.milk() # get the milk
```

BACK TO THE FILE OBJECT

```python
# create an instance of a file object
fo=open("/Users/student/Desktop/python/human-genes.gtf","
    r")
# fo is one instance of the file object

cowreader=open("/Users/student/Desktop/cow-genes.gtf","r"
    )
# cowreader is another instance of the file object

# what are the processing routines (=methods)?
fo.read()
fo.write()
fo.seek()

# Reminder: main use of the file object
for line in fo:
    # do something
```

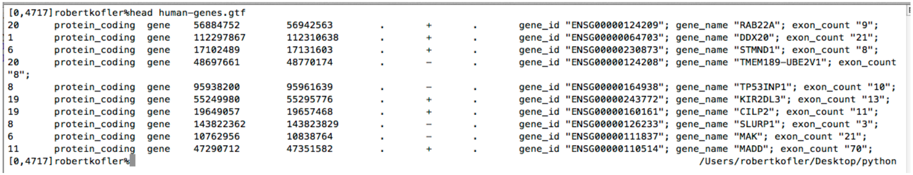
SUMMARY: METHOD VS FUNCTION

- function(): standalone; e.g.: print(), range(), int()
- method(): they are never alone, always only with an instance of an object; e.g.: resi.feed(2), fileobject.read(), string.split(” ”)
- what about: fileobject.feed(2)?

INSPECTING THE HUMAN-GENE FILE

The human genes are stored in the widely used gtf-format.
New UNIX commands "head" and "tail"
Head prints the first 10 lines of a file whereas "tail" prints the last 10 lines of a file.

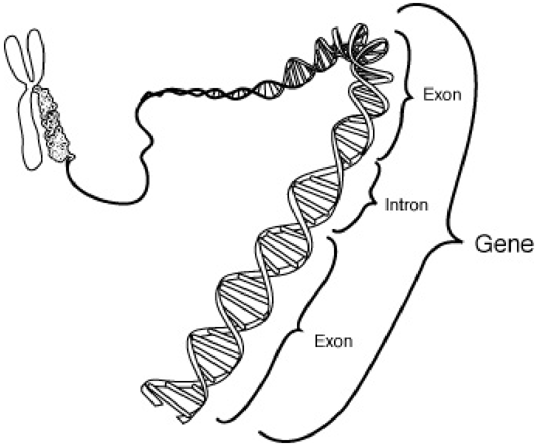
- move into the folder where you stored the gene file!
- type: head human-genes.gtf



```
[0,4717]robertkofler%head human-genes.gtf
20      protein_coding  gene    56884752        56942563        .       +       .       gene_id "ENSG00000124209"; gene_name "RAB22A"; exon_count "9";
1       protein_coding  gene    112297867       112310638       .       +       .       gene_id "ENSG00000064703"; gene_name "DDX20"; exon_count "21";
6       protein_coding  gene    17102489        17131603        .       +       .       gene_id "ENSG00000230873"; gene_name "STMND1"; exon_count "8";
20      protein_coding  gene    48697661        48770174        .       -       .       gene_id "ENSG00000124208"; gene_name "TMEM189-UBE2V1"; exon_count
"8";
8       protein_coding  gene    95938200        95961639        .       -       .       gene_id "ENSG00000164938"; gene_name "TP53INP1"; exon_count "10";
19      protein_coding  gene    55249980        55295776        .       +       .       gene_id "ENSG00000243772"; gene_name "KIR2DL3"; exon_count "13";
19      protein_coding  gene    19649057        19657468        .       +       .       gene_id "ENSG00000160161"; gene_name "CILP2"; exon_count "11";
8       protein_coding  gene    143822362       143823829       .       -       .       gene_id "ENSG00000126233"; gene_name "SLURP1"; exon_count "3";
6       protein_coding  gene    10762956        10838764        .       -       .       gene_id "ENSG00000111837"; gene_name "MAK"; exon_count "21";
11      protein_coding  gene    47290712        47351582        .       +       .       gene_id "ENSG00000110514"; gene_name "MADD"; exon_count "70";
[0,4717]robertkofler%                                                                                                   /Users/robertkofler/Desktop/python
```

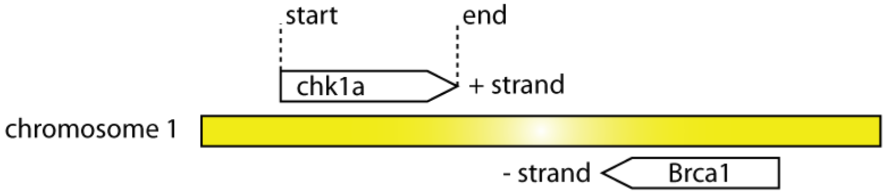
The gtf file contains information about gene's

REFRESHER: GENES AND CHROMOSOMES



A gene is a region in a chromosome, usually codeing for a protein.

Gene in the gtf-file



- column 1: the chromosome of the gene (is a string)
- column 4: start position of the gene
- column 5: end position of the gene
- column 7: strand of the gene
- column 9: diverse information and comments, like the name of the gene

THE LENGTH OF GENES

```python
genepath="/Users/robertkofler/Desktop/python/human-genes.gtf"

# we use a for-loop to iterate over the file, line by line
for line in open(genepath,"r"):
    tmp=line.split("\t") # \t = symbol for tabulator
    start=int(tmp[3])
    end=int(tmp[4])
    print(end-start)

# save as genelength.py -> F5
# press Ctrl - c to stop it
```

SPLIT(): STRING IS ACTUALLY AN OBJECT

```
# create a string
test="the small cat jumps the quick"

# split it at space and we get a list
a=test.split(" ")
print(a)
# NOTE: a list/tuple can contain anything!!
# not just integers

# now we split it at cat
b=test.split("cat")
print(b)

# save as stringobject.py
# F5
```

Special symbols in a string

```
# create a string
test="the\tsmall\ncat\njumps\nthe\tquick\n\n\n"

print(test)
# \t =tabulator
# \n =new line

# rstrip() removes something from the right end
# we want to get rid of the annoying \n\n\n
test=test.rstrip("\n")
print(test)
```

MINITASKS - FIND THE MOST EXTREME GENES

- find the longest gene, print the whole gtf-entry (a single line)
- find both the longest and the shortest gene; print a single line for each

Quick detour: Gene printing improved

```
genepath="/Users/robertkofler/Desktop/python/human-genes.gtf"

# we use a for-loop to iterate over the file, line by line
for line in open(genepath,"r"):
    # when reading a file every
    # line ends with an "\n"  -> so let's remove it
    line=line.rstrip("\n")
    print(line)
```

Task: X vs Y

```
# Task6: X vs Y
# Girls: count the number of genes on the X
    chromosome
# Boys: count the number of genes on the Y
    chromosome
# example output:

# X has ??? genes (girls)
# Y has ??? genes (boys)

# Hint: use split
# save as task6-XvsY.py in your tasks folder
# F5
```

I WAS AT A SOFT SKILL COURSE

About teaching and giving lectures..

ONE INTERESTING INFORMATION

If students explain problems to each other, they both benefit.

- the explaining student: explaining is actually the next level of understanding; this helps to think about the problem more deeply and thus increases competence
- the questioning student: understanding complex problems is obviously easier if explained by someone at a similar level of understanding

$\Rightarrow$ please help each other during tasks

It's about time: our own functions

Which functions do we already know?

- print()
- input()
- int()
- randint()

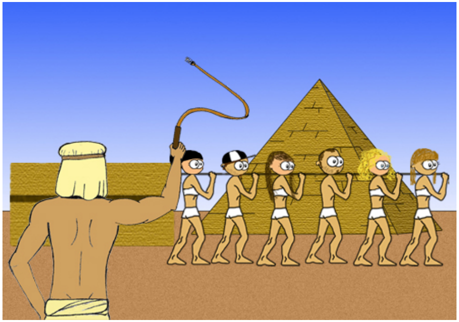
What about mystring.split(" ") ??

A REAL-LIFE PROBLEM FROM A STUDENT JOB

You are working for a Pharaoh, an annoying Pharaoh. He want's to build one pyramid after the other and your job, as his summer intern, is to calculate the number of required stones for every pyramid!!

INTRODUCING: YOUR NEW BOSS

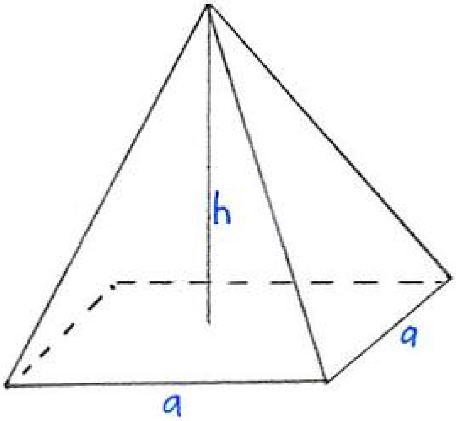
Your new boss intensily studied the art of motivating his employees. After watching plenty of web seminars he suddenly came to a groundbreakingly simple solution:



His revolutionary technique also managed to motivate you beyond your wildest expectations. To avoid even further motivation you decided to solve the problem once and for all by writing a Python script that calculates the number of necessary stones.

In case someone forgot..

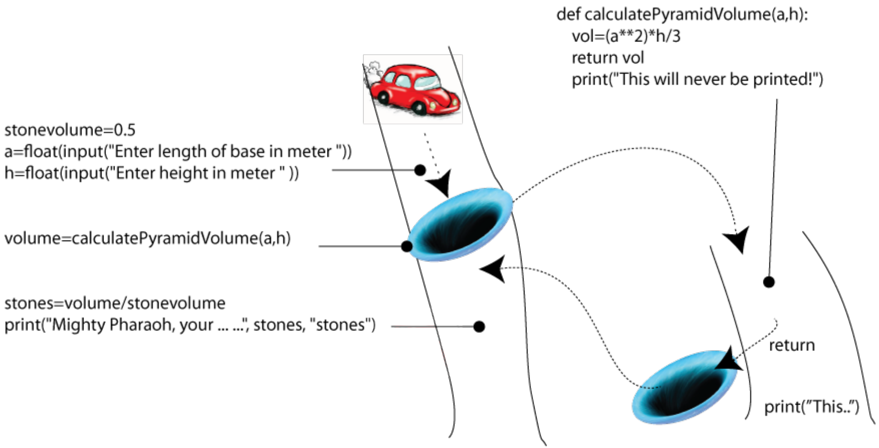
$$V = \frac{a^2 * h}{3}$$



PYRAMID CALCULATOR

```
# first define the function
def calculatePyramidVolume(a,h):
    vol=(a**2)*h/3  # ** power operator
    return vol
    print("This will never be printed!")


# volume of a single stone in m^3
stonevolume=0.5

a=float(input("Enter length of base in meter "))
h=float(input("Enter height in meter " ))

volume=calculatePyramidVolume(a,h)
stones=volume/stonevolume
print("Mighty Pharaoh, your pyramid requires", stones, "stones")

# safe as pyramid.py
# F5
```

CODE FLOW WITH FUNCTION

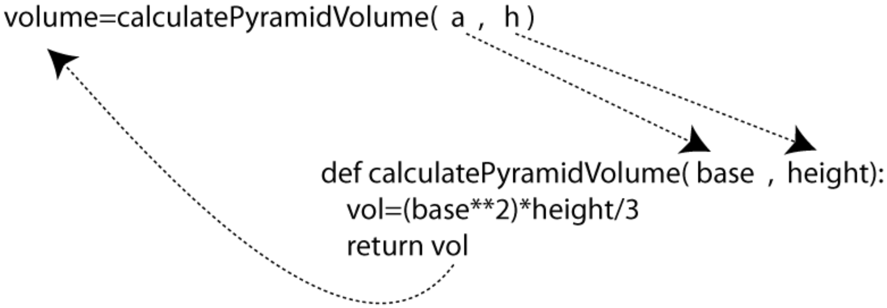


```
def calculatePyramidVolume(a,h):
    vol=(a**2)*h/3
    return vol
    print("This will never be printed!")
```

```
stonevolume=0.5
a=float(input("Enter length of base in meter "))
h=float(input("Enter height in meter " ))

volume=calculatePyramidVolume(a,h)

stones=volume/stonevolume
print("Mighty Pharaoh, your ... ...", stones, "stones")
```

'return' means: jump into the black hole and bring me back to my original road. Everything after return is unreachable!

PARAMETER PASSING FOR FUNCTIONS



```
volume=calculatePyramidVolume( a , h )

def calculatePyramidVolume( base , height):
    vol=(base**2)*height/3
    return vol
```

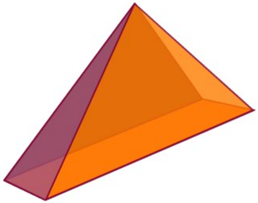
Note: first parameter is passed to first parameter in the function, second to second... the names of the parameters is not important only the order.
Note: the return value is sent back

LET'S AGAIN LOOK AT THE PYRAMID CALCULATOR

```
# first define the function
def calculatePyramidVolume(a,h):
    vol=(a**2)*h/3  # ** power operator
    return vol
    print("This will never be printed!")


# volume of a single stone in m^3
stonevolume=0.5

a=float(input("Enter length of base in meter "))
h=float(input("Enter height in meter " ))

volume=calculatePyramidVolume(a,h)
stones=volume/stonevolume
print("Mighty Pharaoh, your pyramid requires", stones, "stones")

# safe as pyramid.py
# F5
```

MINITASK - RECTANGULAR BASIS

$\Rightarrow$ update the pyramid calculator for pyramids with a rectangular basis



LISTS EXPANDED

```
# create an empty list
lis=[]
print(lis)

# append something to the list
lis.append(2)
lis.append("hallo")
print(lis)
lis.append(3.14159)
print(lis)

# length of list len()
print(len(lis))
# save as listappend.py
# F5
```

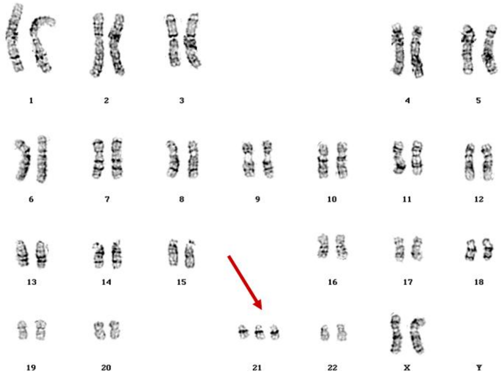
EXAMPLE OF FUNCTION: AVERAGE GENE LENGTH

```
def get_mean(lengs):
    gsum=0
    for l in lengs:
        gsum+=l
    # what is 10/3 ?
    average=float(gsum)/float(len(lengs))
    return average

# again obtain the absolute path of your human-genes files
genepath="/Users/robertkofler/Desktop/python/human-genes.gtf"
lengs=[]
for line in open(genepath,"r"):
    tmp=line.split("\t") # \t = symbol for tabulator
    start=int(tmp[3])
    end=int(tmp[4])
    lengs.append(end-start)

average = get_mean(lengs)
print("Average length of a human gene", average)
# save as averagegenelength.py
# F5
```

YOU ARE COLLABORATING WITH A SCIENTIST

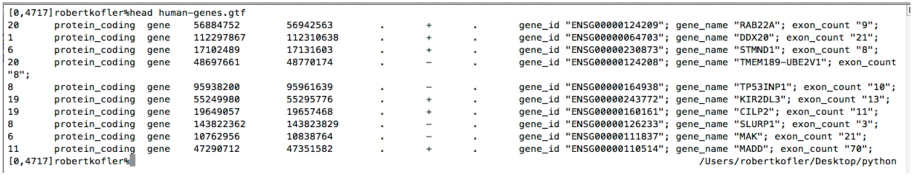
STUDYING TRISOMIE 21 (DOWN-SYNDROME)

The scientist wants to identify genes responsible for the symptomes of Trisomie 21.



To get his work started he needs a list of all the genes on human chromosome 21. So as you are a budding biomedicine expert, he is asking you to send him such a list.

But first: reminder human gene file



```
[0,4717]robertkofler%head human-genes.gtf
20      protein_coding  gene    56884752        56942563        .       +       .       gene_id "ENSG00000124209"; gene_name "RAB22A"; exon_count "9";
1       protein_coding  gene    112297867       112310638       .       +       .       gene_id "ENSG00000064703"; gene_name "DDX20"; exon_count "21";
6       protein_coding  gene    17102489        17131603        .       +       .       gene_id "ENSG00000230873"; gene_name "STMND1"; exon_count "8";
20      protein_coding  gene    48697661        48770174        .       -       .       gene_id "ENSG00000124208"; gene_name "TMEM189-UBE2V1"; exon_count
"8";
8       protein_coding  gene    95938200        95961639        .       -       .       gene_id "ENSG00000164938"; gene_name "TP53INP1"; exon_count "10";
19      protein_coding  gene    55249980        55295776        .       +       .       gene_id "ENSG00000243772"; gene_name "KIR2DL3"; exon_count "13";
19      protein_coding  gene    19649057        19657468        .       +       .       gene_id "ENSG00000160161"; gene_name "CILP2"; exon_count "11";
8       protein_coding  gene    143822362       143823829       .       -       .       gene_id "ENSG00000126233"; gene_name "SLURP1"; exon_count "3";
6       protein_coding  gene    10762956        10838764        .       -       .       gene_id "ENSG00000111837"; gene_name "MAK"; exon_count "21";
11      protein_coding  gene    47290712        47351582        .       +       .       gene_id "ENSG00000110514"; gene_name "MADD"; exon_count "70";
[0,4717]robertkofler%                                                                                                   /Users/robertkofler/Desktop/python
```

- column 1: the chromosome of the gene (is a string)
- column 4: start position of the gene
- column 5: end position of the gene
- column 7: strand of the gene
- column 9: diverse information and comments, like the name of the gene

The information that we need is deeply burried in column 9. How do we get the gene name out of this mess?

GET SUBSTRINGS

```
s="Hello World"
print(s[0]) # print first
print(s[-1]) # print last
print(s[10]) # print last
print(s[-2]) # print what?

# create substrings with s[from:to]
print(s[0:2]) # print first two
print(s[:2])  # also print first two
print(s[2:8]) # print characters from 2 to 8
print(s[-2:]) # print last two
# if from is empty -> default from start of string
# if to is empty -> default to end of string

print(s[1:])  # print what?
print(s[:-1]) # print what?
print(s[1:-1]) # print what?

# the same syntax also works for LISTS/TUPLES
# safe as substring.py -> F5
```

GET GENES OF CHROMOSOME 21

```
def get_genename(comment):
    # example of a comment:
    # gene_id "ENSG00000242220"; gene_name "TCP10L"; exon_count
    "8";
    tmp=comment.split(" ")
    genename=tmp[3]
    genename=genename[1:-2] # what is this doing?
    return genename

genepath="/Users/robertkofler/Desktop/python/human-genes.gtf"
gnames=[]
for line in open(genepath,"r"):
    line=line.rstrip("\n")
    tmp=line.split("\t")
    # col.: 0 = chromosome  col.: 8 = comment
    if(tmp[0]=="21"):
        genename=get_genename(tmp[8])
        gnames.append(genename)

for gn in gnames:
    print(gn)
# save as trisomie21.py -> F5
```

RUN THE SCRIPT FROM THE UNIX-SHELL

- open the command line (terminal)
- cd Desktop/pythonlecture ↩
- in case your folder has a different name use "ls -l" and "cd" to navigate into the folder containing your python script
- python3.3 trisomie21.py > genesofchromosome21.txt ↩
- open this file with a texteditor of your choice (eg. Word)

">" is the redirect operator; You are redirecting the output from the console into a file. The output from the screen is thus saved in the file.

Task 7: X vs Y - round 2

Now that we know the truth about the gene numbers of X and Y it is my hope that the Y at least has ”cooler” genes than the X. Cooler means genes with more exons. This would allow more complex gene products due to alternative splicing.

```
# Task7: X vs Y
# Girls: compute the average number of exons on the X chromosome
# Boys: compute the average number of exons on the Y chromosome
# example output:

# A gene on X has on average ??? exons (girls)
# A gene on Y has on average ??? exons (boys)

# Hint: use two functions
# Hint: one function for getting the exon count
# Hint: one function for calculating the mean
# save as task7-exonsXvsY.py in your tasks folder
# F5
```

THE SCIENTIST CALLED AGAIN

THE PROBLEM

”Some with Down Syndrome are diagnosed with self-regressing TMD and 20-30% of those progress to AMKL. We have performed exome sequencing in 7 cases and copy number analysis in these and 10 additional cases. All TMD/AMKL samples contained GATA1 mutations. None of 7 exome-sequenced TMD/AMKL samples had any other recurrently mutated genes. However 2 of 5 TMD cases, and all sequenced AMKL cases, showed mutations/deletions other than GATA1, in genes proven as transformation-drivers in non-DS (EZH2, APC, FLT3, JAK1, PARK2-PACRG, EXT1, DLEC1 and SMC3).”

After half an hour..

"...Later in infancy, progression requires third-hit-driver mutations/SCNAs found in non-DS. Putative driver-mutations affecting WNT, JAK-STAT or MAPK/PI3K pathways were found in all cases, aberrant activation of which converges on overexpression of MYC"
Translation: "The gene in which I'm interested is within 15Mbp and 25Mbp in chromosome 21, please sent me a new list of the genes in this region.

SUB-REGION OF CHROMOSOME 21

```
def get_genename(comment):
    tmp=comment.split(" ")
    genename=tmp[3]
    genename=genename[1:-2]
    return genename

genepath="/Users/robertkofler/Desktop/python/human-genes.gtf"
gnames=[]
for line in open(genepath,"r"):
    line=line.rstrip("\n")
    tmp=line.split("\t")
    startpos=int(tmp[3])
    endpos=int(tmp[4])
    # col.: 0 = chromosome  col.: 8 = comment
    if(tmp[0]=="21" and startpos > 15000000 and  endpos <
     25000000):
        genename=get_genename(tmp[8])
        gnames.append(genename)

for gn in gnames:
    print(gn)
# save as trisomie21update.py -> F5
```

AND,OR,NOT

- and: both statements need to be true
- or: one of the statements need to be true
- not: the statement needs to be false

AND,OR,NOT EXAMPLES

```python
# and examples
number = 21
print("and example:", number > 10 and number < 30)
print("and example:", number > 10 and number <20)

# or examples
print("or example:",number > 10 or number < 20)
print("or example:",number > 25 or number < 20)

# not examples
print("not example:", number >10)
print("not example:", not number > 10)

# what will happen here?
print(not number < 10)
print(number < 42 or number > 30)
print((not number <10) and number <30)
```

We send him this new list of genes

So we sent him this new list of genes and he is happy!!!
......well.......
...for 1 day.....
....than he calls again:

- my gene is between 20Mbp and 25Mbp
- no wait I think it is between 25Mbp and 30Mbp
- or maybee I'm entirely wrong and it is within 30 and 45Mbp, hmm??
- ....

You are getting quite annoyed and suspect that this will be a never ending story. What to do, any ideas?

THE SOLUTION..

- He should run the script himself, where he can enter the lower and the upper boundary of the genes of interest as parameters.
- So we will have to write a script for our scientist that can be ”easily” used.
- Easily means from the command line..
- This will be the first script that you can not run in IDLE! The script requires parameters entered in the comand line for execution.

Scientist satisfied

```python
import sys

def get_genename(comment):
    tmp=comment.split(" ")
    genename=tmp[3]
    genename=genename[1:-2]
    return genename

lower=int(sys.argv[1])
upper=int(sys.argv[2])
genepath="/Users/robertkofler/Desktop/python/human-genes.gtf"
gnames=[]
for line in open(genepath,"r"):
    line=line.rstrip("\n")
    tmp=line.split("\t")
    startpos=int(tmp[3])
    endpos=int(tmp[4])
    # col.: 0 = chromosome  col.: 8 = comment
    if(tmp[0]=="21" and startpos > lower and  endpos < upper):
        genename=get_genename(tmp[8])
        gnames.append(genename)

for gn in gnames:
    print(gn)
# save as trisomie21cl.py -> F5
```

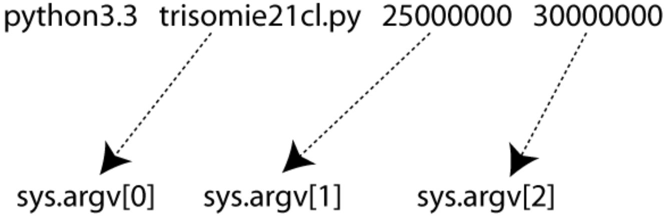
Run the script from the command line

- open the command line (terminal)
- cd Desktop/pythonlecture ↵
- in case your folder has a different name use "ls -l" and "cd" to navigate into the folder containing your python script
- python3.3 trisomie21cl.py 25000000 30000000 ↵
- ↑
- python3.3 trisomie21cl.py 25000000 30000000 > genelist.gtf ↵

↑ is fetching the previous command
Remember: > means redirect into file.

SYS.ARGV?

Is a predifined list that contains all parameters provided by the user.



LET'S INSPECT THE SCRIPT AGAIN

```
import sys

def get_genename(comment):
    tmp=comment.split(" ")
    genename=tmp[3]
    genename=genename[1:-2]
    return genename

lower=int(sys.argv[1])
upper=int(sys.argv[2])
genepath="/Users/robertkofler/Desktop/python/human-genes.gtf"
gnames=[]
for line in open(genepath,"r"):
    line=line.rstrip("\n")
    tmp=line.split("\t")
    startpos=int(tmp[3])
    endpos=int(tmp[4])
    # col.: 0 = chromosome  col.: 8 = comment
    if(tmp[0]=="21" and startpos > lower and  endpos < upper):
        genename=get_genename(tmp[8])
        gnames.append(genename)

for gn in gnames:
    print(gn)
# save as trisomie21cl.py -> F5
```

MINITASK - EXPAND THE SCRIPT FOR THE SCIENTIST

```
# Remember the script where the scientist had to enter
# the minimum and the maximum position of a gene on
# chromosome 21 in the command line like:

# python3.3 trisomie21cl.py 25000000 30000000

# now he wants to have an improved script where he can
    additionaly
# also enter the minimum and the maximum length of a gene on
# the command line like:

# python3.3 task8-trisomie21.py 15000000 25000000 10000 100000


#run it on the command line with 15000000 25000000 10000 100000
# Crosscheck: this should yield 8 genes!
```

A BRILLIANT IDEA??

Guess what, the scientist called again!

"I have a brilliant idea, I think I know the reason why chromosome 21 is the only chromosome where three copies in the genome do not lead to instant death. It is because it has the fewest genes!! Can you please help me and test if my hypothesis is correct??"

Another script for the scientist??

You really do not want to write another script for the scientist..

However as he is looking at you in a very strange way you decided to help him...again..

Note: if you decide not to learn programming it may be favorable for your carreer if you train this expression instead ;)

BUT FIRST...INTRODUCING HASH/DICTIONARY

We have a new data structure it is called hash or dictionary.

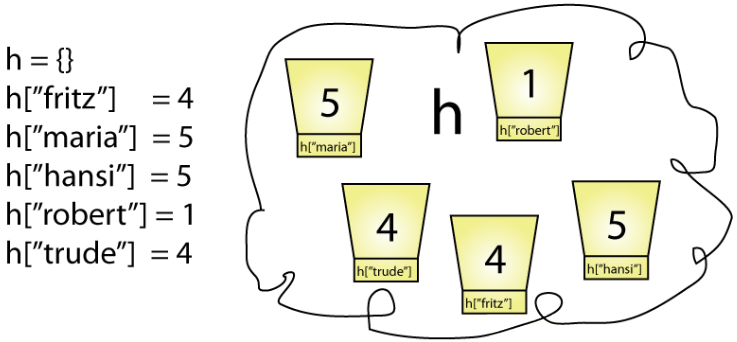
```
# appologies this is a dating planer for guys
# Ladies you can use guy names if you want!

names={} # curly brackets creates an empty dictionary
names["babsi"] = True    # True means single
names["maria"] = False   # False means married
names["eva"] = True

todate=input("Who’m do you want to date? ")
# the in command asks whether such a name is in the hash
if(todate in names):
    if(names[todate]==True):
        print("Great she is single, you can date her")
    else:
        print("Well she is married, but heck, you can date her")
else:
    print("Sorry I have no information about",todate)
    print("But this should not stop you from dating her")

# save as datingplaner.py -> F5
```

A hash consists of key-value pairs

A hash is similar to a list. The key has to be a unique identifier. Values are any information somehow connected to the key. The key has to be immutable (string, tuple, integer, (list?)) whereas the value may be anything (immutable or mutable).
Examples of key/value pairs:

- Name / address
- Student ID / mark in exam
- Patient ID / medical condition
- Chromosome name / genes on chromosome
- ...?

RESULTS OF A TYPICAL EXAM IN "INTRODUCTION TO PROGRAMMING"



```
h = {}
h["fritz"]   = 4
h["maria"]  = 5
h["hansi"]  = 5
h["robert"] = 1
h["trude"]  = 4
```

NOTE: In a hash, there is no first or last element (compare to list, which is the first which the last element?). All elements are equally accessible via the key.

KEY, VALUE, ITEM

How to access the elements of a hash?

```
# Dateingplaner age
# introudces key() value() items()

names={} # curly brackets creates an empty dictionary
names["babsi"] = 19
names["agnes"] = 31
names["maria"] = 21
names["eva"] = 25 # only legal ages in the dateing planer

print(names.keys())
print(names.values())
print(names.items())
# -> F5
```

COMPLEX DATA STRUCTURE

```python
# Complex data structure

# a list of lists
lol=[[1,2,3],[5,7,9]]

# a tuple of tuples
tot=(("l","i","s"),("t"),("of", "tuples"))

# a list of tuples
lot=[('marie',20),('agnes',32),('babsi',19),('miazi',70)]

# chaoslist
chaos=["this",("i","s"),"a",["tot","al"],'mess']

print(lol)
print(tot)
print(lot)
print(chaos)
# F5
```

MINITASK DATING

```python
# Mini task dateing
lot=[('marie',20),('agnes',32),('babsi',19),('miazi',70),('sofie'
    ,34)]

# print a list of names ang ages like
# marie 20
# agnes 32
# babsi 19
# ...
# F5
```

MINITASK CREATE A COMPLEX DATA STRUCTURE

```
# Mini task
# create a complex data structure with loops
# a list of lists containing
# [[1, 2, 3, 4, 5], [2, 4, 6, 8, 10],
# [3, 6, 9, 12, 15], [4, 8, 12, 16, 20], [5, 10, 15, 20, 25]]

# Hints
# sub lists start with 1,2,3,4,5
# values within sub-lists are multiplied by 1,2,3,4,5
```

SORTING OF LISTS

```
# sorting
l=(5,4,2,10,12)
print(l)

# sorting
l=sorted(l)
print(l)

# reversing
l=list(reversed(l))
print(l)

names=('marie','holly','anna','babs')
print(sorted(names))
```

Sorting of complex data structures

```
# how to sort complex data types?
lot=[('marie',20),('agnes',32),('babsi',19),('miazi',70),('sofie'
    ,34)]

# sorted by name
lot=sorted(lot)
print(lot)

# but how to sort by age??
age=sorted(lot,key=lambda i: i[1])
print(age)
# a lambda expression.....
# is a shortcut for writting a function


def sortorder(i):
    return i[1]

resort=sorted(lot,key=sortorder)
print(resort)
```

TASK - SO IS THE SCIENTIST RIGHT?

```
# Task gene count per chromosome
# Print for every chromosome the number of genes
# starting with the chromosome having the fewest genes
# ending with the chromosome having the most genes
#
# Output should look like this:
# Chromosome xx has 57 genes
# Chromosome xx has 245 genes
# Chromosome xx has 289 genes
# Chromosome xx has 331 genes
# Chromosome xx has 459 genes
# .....
# Hints:
# a) use a dictionary to count the genes
# b) get the items and sort
# safe as Task-genecount.py
```