Python Programming for Life Scientists

Boas Pucker

Introduction of participants

- Name
- Study program (semester)
- Bioinformatics experiences
- Research interests
- motivation?
- …

Boas Pucker

2009-2010: Biochemistry

2010-2013: Biology (Genetics, Cell Biology & Physiology)
Functional Genomics, Metabolic Engineering, Comparative Genomics

2013-2015: Genome-based Systems Biology
Synthetic Biology (iGEM2014), Software Development, Genomics

Since 2016: PhD student in Genome Research & Bioinformatics
Plant Genomics, Bioinformatics, Synthetic Biology
(iGEM2016/2017/2018)

2018: Visiting Scientist at the Brockington Lab, Plant Sciences,
University of Cambridge

Research

Gene prediction

Functional annotation

De novo genome assembly

De novo transcriptome assembly

synteny

Genome Research

RNA-Seq analysis

phylogeny

Co-expression analysis

…

Gene families / pathways

- Public scripts: https://github.com/bpucker

Example documentation

```
<html>

    <head>
        <h1>Applied Python Programming for Life Scientists</h1>
        <!-- fill in the following fields -->
        <h2>SEMESTER</h2>
        <h2>NAME</h2>
        <h2>E-MAIL</h2>
    </head>


    <body>


        <!-- here starts one block -->
        <div id="day1">
            <h3>Day 1 - Introduction</h3>
            <p>all important inforamtion</p>
            <p>homework: request UNIX-Account at CeBiTec Support</p>
        </div>
        <!-- here ends one block (can be copied) -->



    </body>

</html>
```

Aims & content

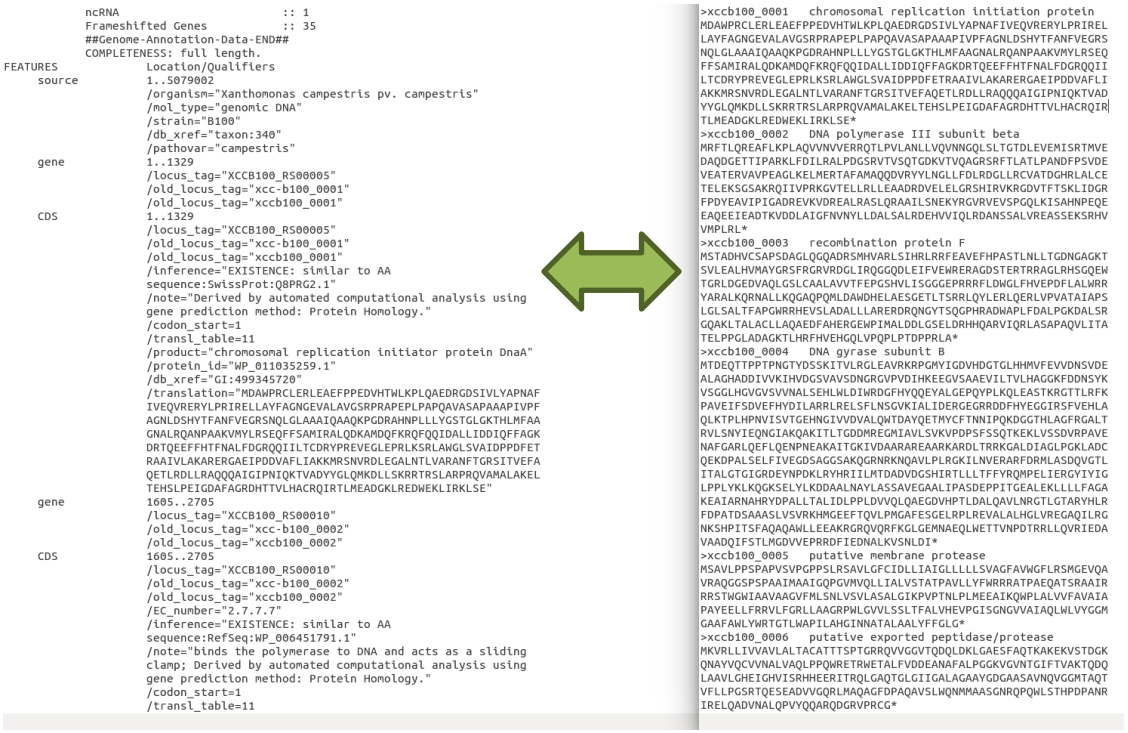
- Basics of Python syntax


```
1  def print_hello_world():
2      """ function prints 'hello world!' """
3
4      print 'hello world!'
5
6
7  print_hello_world() #calling function
8
```

Aims & content

- Basics of Python syntax
- parsing, filtering, exporting, converting

Genbank (.gb)



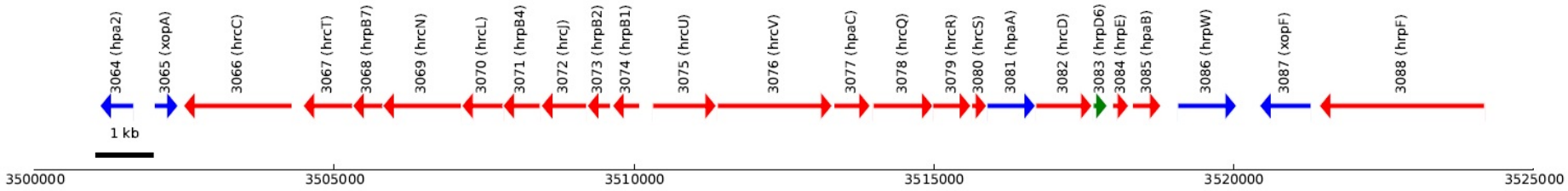
```
          ncRNA                     :: 1
          Frameshifted Genes        :: 35
          ##Genome-Annotation-Data-END##
          COMPLETENESS: full length.
FEATURES             Location/Qualifiers
     source          1..5079002
                     /organism="Xanthomonas campestris pv. campestris"
                     /mol_type="genomic DNA"
                     /strain="B100"
                     /db_xref="taxon:340"
                     /pathovar="campestris"
     gene            1..1329
                     /locus_tag="XCCB100_RS00005"
                     /old_locus_tag="xcc-b100_0001"
                     /old_locus_tag="xccb100_0001"
     CDS             1..1329
                     /locus_tag="XCCB100_RS00005"
                     /old_locus_tag="xcc-b100_0001"
                     /old_locus_tag="xccb100_0001"
                     /inference="EXISTENCE: similar to AA
                     sequence:SwissProt:Q8PRG2.1"
                     /note="Derived by automated computational analysis using
                     gene prediction method: Protein Homology."
                     /codon_start=1
                     /transl_table=11
                     /product="chromosomal replication initiator protein DnaA"
                     /protein_id="WP_011035259.1"
                     /db_xref="GI:499345720"
                     /translation="MDAWPRCLERLEAEFPPEDVHTWLKPLQAEDRGDSIVLYAPNAF
                     IVEQVRERYLPRIRELLAYFAGNGEVALAVGSRPRAPEPLPAPQAVASAPAAAPIVPF
                     AGNLDSHYTFANFVEGRSNQLGLAAAIQAAQKPGDRAHNPLLLYGSTGLGKTHLMFAA
                     GNALRQANPAAKVMYLRSEQFFSAMIRALQDKAMDQFKRQFQQIDALLIDDIQFFAGK
                     DRTQEEFFHTFNALFDGRQQIILTCDRYPREVEGLEPRLKSRLAWGLSVAIDPPDFET
                     RAAIVLAKARERGAEIPDDVAFLIAKKMRSNVRDLEGALNTLVARANFTGRSITVEFA
                     QETLRDLLRAQQQAIGIPNIQKTVADYYGLQMKDLLSKRRTRSLARPRQVAMALAKEL
                     TEHSLPEIGDAFAGRDHTTVLHACRQIRTLMEADGKLREDWEKLIRKLSE"
     gene            1605..2705
                     /locus_tag="XCCB100_RS00010"
                     /old_locus_tag="xcc-b100_0002"
                     /old_locus_tag="xccb100_0002"
     CDS             1605..2705
                     /locus_tag="XCCB100_RS00010"
                     /old_locus_tag="xcc-b100_0002"
                     /old_locus_tag="xccb100_0002"
                     /EC_number="2.7.7.7"
                     /inference="EXISTENCE: similar to AA
                     sequence:RefSeq:WP_006451791.1"
                     /note="binds the polymerase to DNA and acts as a sliding
                     clamp; Derived by automated computational analysis using
                     gene prediction method: Protein Homology."
                     /codon_start=1
                     /transl_table=11
```

```
>xccb100_0001   chromosomal replication initiation protein
MDAWPRCLERLEAEFPPEDVHTWLKPLQAEDRGDSIVLYAPNAFIVEQVRERYLPRIREL
LAYFAGNGEVALAVGSRPRAPEPLPAPQAVASAPAAAPIVPFAGNLDSHYTFANFVEGRS
NQLGLAAAIQAAQKPGDRAHNPLLLYGSTGLGKTHLMFAAGNALRQANPAAKVMYLRSEQ
FFSAMIRALQDKAMDQFKRQFQQIDALLIDDIQFFAGKDRTQEEFFHTFNALFDGRQQII
LTCDRYPREVEGLEPRLKSRLAWGLSVAIDPPDFETRAAIVLAKARERGAEIPDDVAFLI
AKKMRSNVRDLEGALNTLVARANFTGRSITVEFAQETLRDLLRAQQQAIGIPNIQKTVAD
YYGLQMKDLLSKRRTRSLARPRQVAMALAKELTEHSLPEIGDAFAGRDHTTVLHACRQIR
TLMEADGKLREDWEKLIRKLSE*
>xccb100_0002   DNA polymerase III subunit beta
MRFTLQREAFLKPLAQVVNVVERRQTLPVLANLLVQVNNGQLSLTGTDLEVEMISRTMVE
DAQDGETTIPARKLFDILRALPDGSRVTVSQTGDKVTVQAGRSRFTLATLPANDFPSVDE
VEATERVAVPEAGLKELMERTAFAMAQQDVRYYLNGLLFDLRDGLLRCVATDGHRLALCE
TELEKSGSAKRQIIVPRKGVTELLRLLEAADRDVELELGRSHIRVKRGDVTFTSKLIDGR
FPDYEAVIPIGADREVKVDREALRASLQRAAILSNEKYRGVRVEVSPGQLKISAHNPEQE
EAQEEIEADTKVDDLAIGFNVNYLLDALSALRDEHVVIQLRDANSSALVREASSEKSRHV
VMPLRL*
>xccb100_0003   recombination protein F
MSTADHVCSAPSDAGLQGQADRSMHVARLSIHRLRRFEAVEFHPASTLNLLTGDNGAGKT
SVLEALHVMAYGRSFRGRVRDGLIRQGGQDLEIFVEWRERAGDSTERTRRAGLRHSGQEW
TGRLDGEDVAQLGSLCAALAVVTFEPGSHVLISGGGEPRRRFLDWGLFHVEPDFLALWRR
YARALKQRNALLKQGAQPQMLDAWDHELAESGETLTSRRLQYLERLQERLVPVATAIAPS
LGLSALTFAPGWRRHEVSLADALLLARERDRQNGYTSQGPHRADWAPLFDALPGKDALSR
GQAKLTALACLLAQAEDFAHERGEWPIMALDDLGSELDRHHQARVIQRLASAPAQVLITA
TELPPGLADAGKTLHRFHVEHGQLVPQPLPTDPPRLA*
>xccb100_0004   DNA gyrase subunit B
MTDEQTTPPTPNGTYDSSKITVLRGLEAVRKRPGMYIGDVHDGTGLHHMVFEVVDNSVDE
ALAGHADDIVVKIHVDGSVAVSDNGRGVPVDIHKEEGVSAAEVILTVLHAGGKFDDNSYK
VSGGLHGVGVSVVNALSEHLWLDIWRDGFHYQQEYALGEPQYPLKQLEASTKRGTTLRFK
PAVEIFSDVEFHYDILARRLRELSFLNSGVKIALIDERGEGRRDDFHYEGGIRSFVEHLA
QLKTPLHPNVISVTGEHNGIVVDVALQWTDAYQETMYCFTNNIPQKDGGTHLAGFRGALT
RVLSNYIEQNGIAKQAKITLTGDDMREGMIAVLSVKVPDPSFSSQTKEKLVSSDVRPAVE
NAFGARLQEFLQENPNEAKAITGKIVDAARAREAARKARDLTRRKGALDIAGLPGKLADC
QEKDPALSELFIVEGDSAGGSAKQGRNRKNQAVLPLRGKILNVERARFDRMLASDQVGTL
ITALGTGIGRDEYNPDKLRYHRIILMTDADVDGSHIRTLLLTFFYRQMPELIERGYIYIG
LPPLYKLKQGKSELYLKDDAALNAYLASSAVEGAALIPASDEPPITGEALEKLLLFAGA
KEAIARNAHRYDPALLTALIDLPPLDVVQLQAEGDVHPTLDALQAVLNRGTLGTARYHLR
FDPATDSAAASLVSVRKHMGEEFTQVLPMGAFESGELRPLREVALALHGLVREGAQILRG
NKSHPITSFAQAQAWLLEEAKRGRQVQRFKGLGEMNAEQLWETTVNPDTRRLLQVRIEDA
VAADQIFSTLMGDVVEPRRDFIEDNALKVSNLDI*
>xccb100_0005   putative membrane protease
MSAVLPPSPAPVSVPGPPSLRSAVLGFCIDLLIAIGLLLLLSVAGFAVWGFLRSMGEVQA
VRAQGGSPSPAAIMAAIGQPGVMVQLLIALVSTATPAVLLYFWRRATPAEQATSRAAIR
RRSTWGWIAAVAAGVFMLSNLVSVLASALGIKPVPTNLPLMEEAIKQWPLALVVFAVAIA
PAYEELLFRRVLFGRLLAAGRPWLGVVLSSLTFALVHEVPGISGNGVVAIAQLWLVYGGM
GAAFAWLYWRTGTLWAPILAHGINNATALAALYFFGLG*
>xccb100_0006   putative exported peptidase/protease
MKVRLLIVVAVLALTACATTTSPTGRRQVVGGVTQDQLDKLGAESFAQTKAKEKVSTDGK
QNAYVQCVVNALVAQLPPQWRETRWETALFVDDEANAFALPGGKVGVNTGIFTVAKTQDQ
LAAVLGHEIGHVISRHHEERITRQLGAQTGLGIIGALAGAAYGDGAASAVNQVGGMTAQT
VFLLPGSRTQESEADVVGQRLMAQAGFDPAQAVSLWQNMMAASGNRQPQWLSTHPDPANR
IRELQADVNALQPVYQQARQDGRVPRCG*
```

FASTA (.fa/.fas/.fasta)

Aims & content

- Basics of Python syntax
- parsing, filtering, exporting, converting
- Generate figures (e.g. matplotlib)

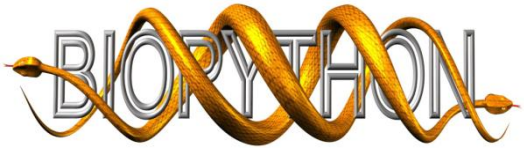


Aims & content

- Basics of Python syntax
- parsing, filtering, exporting, converting
- Generate figures (e.g. matplotlib)
- Own projects/challenges?! => own solution & success

WhyPython?

- Easy to learn
- More efficient than Excel => important for big data
- Check/modify => unlimited opportunities
- Script needs to be written once and can be applied often
  - => frequent problems are ideal for bioinformatics
- Very powerful with many libraries/modules
  - biopython (including NCBI BLAST+), scipy, numpy, Rpy, matplotlib …



Examples

- Everything Excel does and much more
- Primer design and validation
  - Identify binding sites and their distances, nucleotide composition, codon frequency, etc.
- BLAST + evaluation of results
  - *in silico* translation of multiple sequences and automatic inspection of all resulting gene products
- Convert data formats (e.g. FASTA-like > FASTA)
- Advanced search&replace
- You own ideas!!!

Outline

- Environment: ThinLinc, Xterm, Python
- Basic commands and data structures
- functions
- Control structures (if, else, for, while)
- File handling
- …
- Your own project!!!

Homework

- A: Request UNIX-Account at CeBiTec Support (V6-126) if applicable (group: “apbiokurs”)
- B: UNIX acount name to **boas.pucker[at]uni-bielefeld.de** for group invitation
- Collect own bioinformatic challenges for potential projects!

Environment: ThinLinc

ThinLinc

- IGLE or client on own computer
- Download ThinLinc client:
  - https://www.cendio.com/thinlinc/download
- LogIn:
  - Server: thinlinc.cebitec.uni-bielefeld.de
  - ID + PW

xterm / (l)qxterm

- Look for this symbol at bottom of screen
  - => xterm is started by clicking on it

- Type “lqxterm” into the terminal and hit ENTER
  => **always use lqxterm for running Python scripts!**

- (l)qxterms are intended for heavy computation

Starting Python

- Type the following commands into the lqxterm:

  which python

  - => displays default Python location/installation

  python

  - => starts Python: version 2.7.xxx should be displayed
  - Enter this and hit ENTER: print “hello world!”

Starting Python

- Type the following commands into the lqxterm:

  which python
  - => displays default Python location/installation

  python
  - => starts Python: version 2.7.xxx should be displayed
  - Enter this and hit ENTER: print “hello world!”

- Output: hello world!

  => Python is running!

File manager

- Click on “homes”
  => Opens your personal “homes”

- Type“/vol/apbiokurs/members” into file manager and hit ENTER

- right click and create a new subfolder: <UNIX-Name>

- Generate to subdirectories:
  - “scripts”
  - “data”

The first Python script

- Create new text file in directory “scripts” (via right click > new)
- Name it “test.py” (rename via right click on file)
- Right click on file > “open with” > “geany”
- Write “print ‘hello world!’ ” into file and save

Executing Python script

- Select lqxterm:

  CONTROL+ D (press at same time) => Python terminates

- Enter in lqxterm:

  cd /vol/apbiokurs/members/<UNIX-Name>/scripts

- cd (=change directory) sets the current working directory
- Enter in lqxterm:

  python test.py

- What happend?

Space is important!

Executing Python script

- Select lqxterm:

  CONTROL+ D (press at same time) => Python terminates

- Enter in lqxterm:

  cd /vol/apbiokurs/members/<UNIX-Name>/scripts

- cd (=change directory) sets the current working directory
- Enter in lqxterm:

  python test.py

- What happend?

Executing Python script II

- Arrow buttons can be used to navigate through history:
  Arrow up=> last command (hit ENTER to confirm)

- Editing of previous commands (e.g. removal of typos)

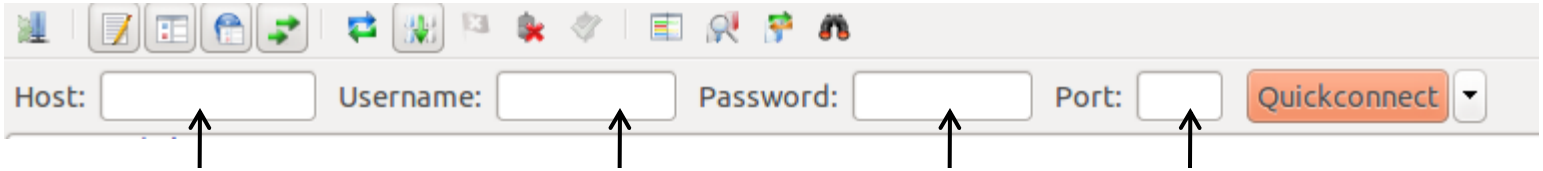
- Important: script is always processed from top to bottom!

Filezilla

- Download:

  http://www.chip.de/downloads/FileZilla_13011076.html

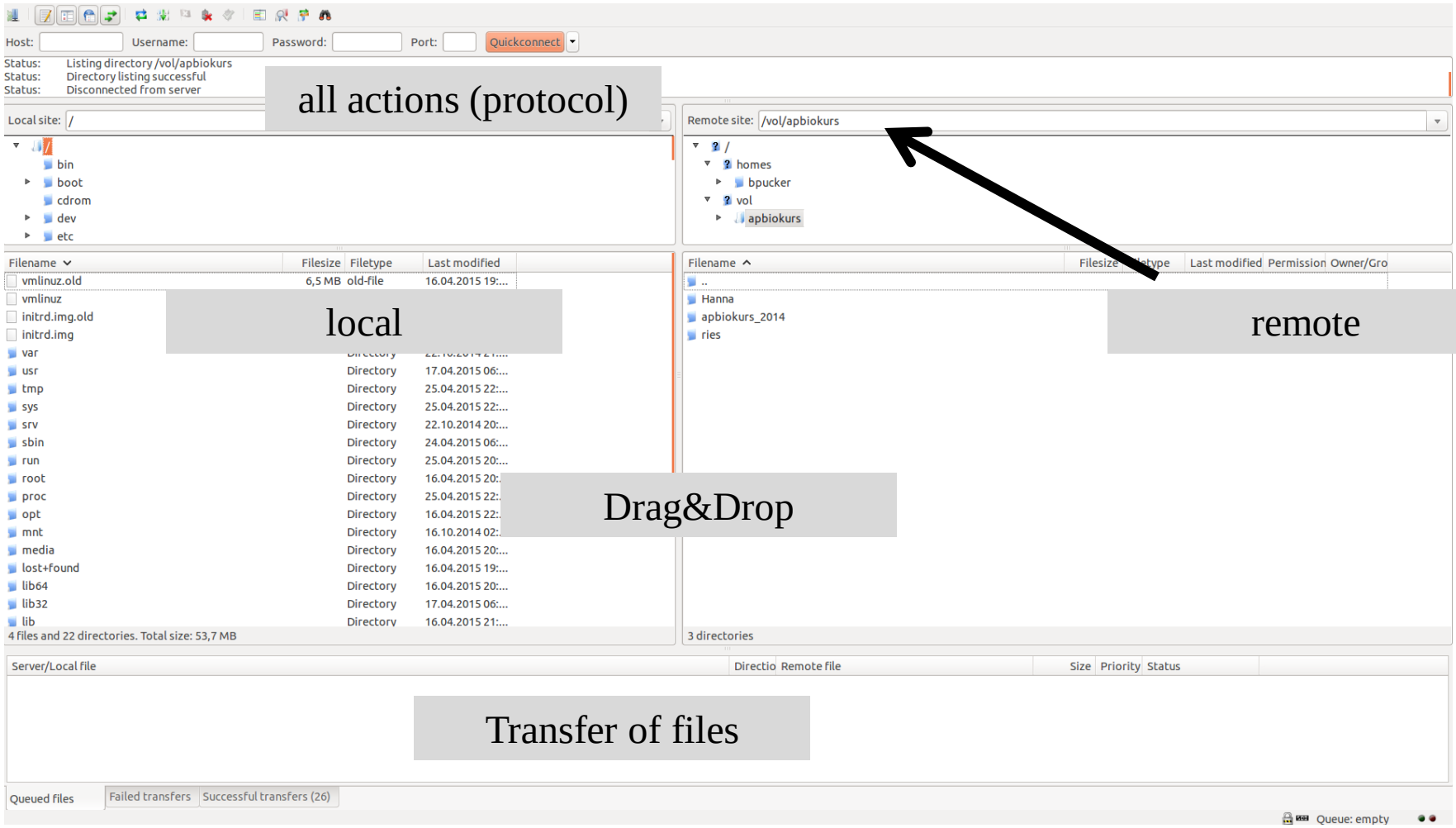
  Installation (more recent version)
- Start Filezilla:



porta.cebitec.uni-bielefeld.de UNIX-ID UNIX-PW 22

- Path to files: /vol/apbiokurs

Filezilla II



Simple commands & variable types

```
1
2  print "test"                                  #print
3  my_list = ["1", "2", "3", 4, 5, 6]           #list
4  my_int = 1                                   #int
5  my_float = float( 3.1 )                      #float
6  my_string = str( my_int )                    #string
7
```

Comments / structure

- To ways to add comments:
  - '#' rest of the line is commend and ignored by Python
  - ' “””” comment “””” ' text in triple quotation marks is a comment which can be extended over multiple lines

- Use ASCII characters only (NO ä, ö, ü, ß …)

- Empty lines are ignored by Python

  => Use space to structure code

Assignment / comparison

- ‘=’ used to assign value to variable:

  a = “hello world!”

  print a

- ‘==’ compares two values/variables:

  a = “hello world!”

  b = “test”

  c = “test”

  a == b

  b == c

- Variable names may contain characters, underline, and numbers (not at the start!)

Variable type string

- a, b, and c are strings (“str”)

- Python allows to check the variable type:

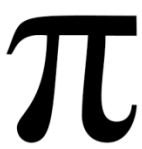
  type(a)

- Almost all variable types can be converted to string:

  str(<VARIABLE>)

Variable types integer & float

- Two variable types for numbers:
  - integer = complete number (example: 3)
  - float = decimal number (example: 3.1415926)

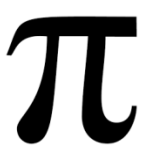


- Important: “.” NOT“,” separates numbers in float!

- Some strings can be converted to integer/float:

  my_int = int(“3”)

  my_float = float(“3.145926”)



- Check result via type(<VARIABLE>)

Python as calculator

- Numbers can be used for calculations ;-)

```
a = 3
b = 2
print a+b          #addition
print a*b          #multiplication
print a**b         #exponentiation
print a/b          #division
print a/float(b)
print a%b          #modulo division
print a < b        #alternatives: >=, <=, >, and ==
print a != b                  #test for inequality
```

- Calculating roots?
- Interested in more complex math? => numpy, scipy

Variable type list

- List can contain elements of different types (e.g. strings):

```
my_list = [ a, b, c ]              #list of strings
print my_list
new_list = [ “eins”, “zwei”, “drei” ]
```

- Elements can be accessed via index
- Index is given in square brackets after the list name:

```
print new_list[ 1 ]
```

- Matching your expectation?

Indices in Python

- Python starts counting at 0!!!

```
new_list = [ “eins”, “zwei”, “drei” ]
#index:          0          1          2
```

- Lists can be concatenated:

```
new_list = new_list + [ “vier”, “fünf”, “sechs”, “sieben” ]
#new_list = [ “eins”, “zwei”, “drei”, “vier”, “fünf”, “sechs”, “sieben” ]
#index =        0          1          2          3          4          5              6
```

- Print subset of list:

```
print new_list[ 3: ]          #elements with index to end
print new_list[:3]            #elements in front of the given index
print new_list[ 3:5 ]         #elements with first index in front of second
```

Indices in Python II

- Strings have indices as well:

```
a = “hello world test string!”
print a[1:]
print a[5:10]
print a[:-1]        #-1 = only last element
print a[-5:-1]      #-5 = starting from 5th element from the end of the
                    #list without the very last one
```

Variable type boolean (True/False)

- Already used for comparison:

```
print 1 == 1
print 1 > 1
print 1 == True
print True+True
print True + False
```

- Boolean variables can be used for calculations (like numbers)

- Most of the time used only for internal calculations

Brackets

- Two important types of brackets:

  [ ] to generate lists and to access elements via index

  ```
  a = [ ]          #empty list
  b = “test”       #a string
  print b[1]
  ```

  () to transfer arguments to functions

- => What are functions?

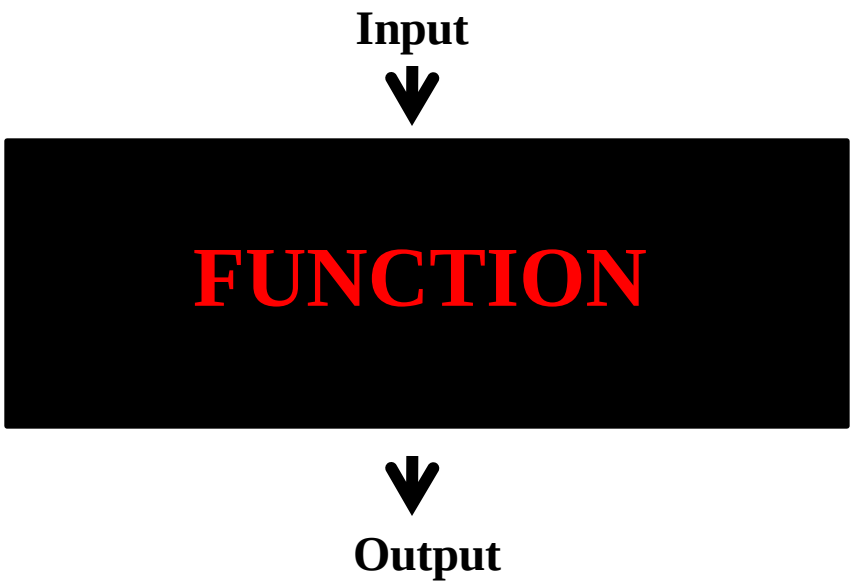
  Examples:

  - str(<VARIABLE>)
  - int(<VARIABLE>)
  - float(<VARIABLE>)

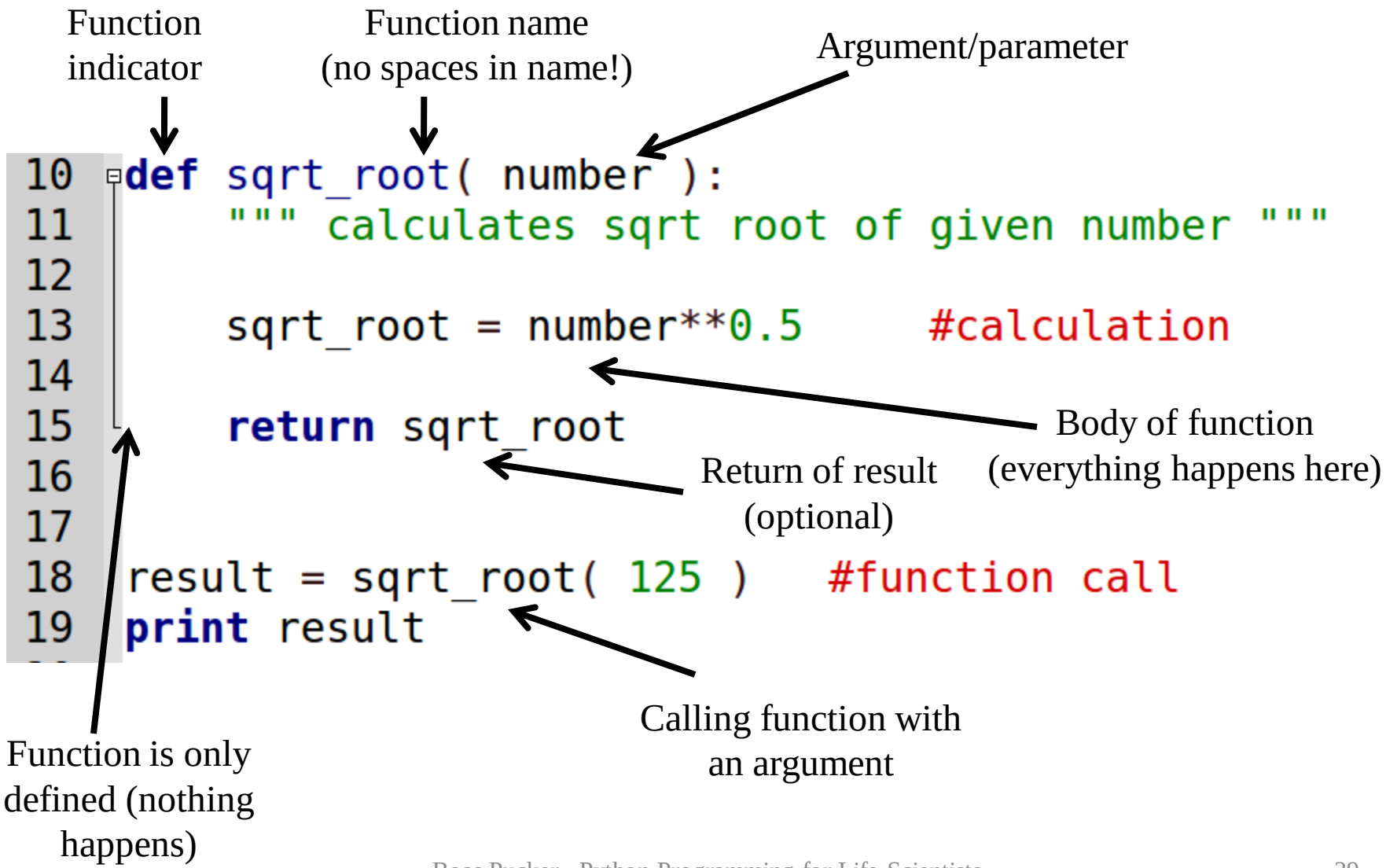
Exercises

- 3.1) Save 3,14159265359 in a variable of type float!
- 3.2) Convert variable from float to integer!
- 3.3) Convert variable back! What happens?
- 3.4) Convert variable type to string!
- 3.5) Save ‘Python’ in a string variable!
- 3.6) Convert variable type to float! What happens?
- 3.7) What is a pitfall in regards to division when working with int/float?

Functions



Functions


Function indicator
Function name (no spaces in name!)
Argument/parameter
10 def sqrt_root(number):
11 """ calculates sqrt root of given number """
12
13 sqrt_root = number**0.5 #calculation
14
15 return sqrt_root
16
17
18 result = sqrt_root(125) #function call
19 print result
Body of function (everything happens here)
Return of result (optional)
Calling function with an argument
Function is only defined (nothing happens)

Advantages of functions

- Generate modules=> write it ones and apply it often (for different purposes)
- structure=> increases readability of your code
- Nesting of calculations:

```
1  def get_gc_content( seq ):
2      """ calculates GC content of given sequence """
3      return float( seq.count('G')+ seq.count('C') ) / len(seq)
4
5  gc_content = get_gc_content("ATGCGACTCAATGCA")
6  print gc_content
```

Important functions

str(<VARIABLE>)	#converts variable to string
int(<VARIABLE>)	#converts variable to integer
float(<VARIABLE>)	#converts variable to float
<STRING1>.count(“<STRING2>”)	#counts occurrences of string2 in string1
<LISTE>.count(<LISTENELEMENT>)	#counts occurrences of element in list
len(<STRING/LISTE>)	#calculates length of string/list

- Warning: Functions return error if invalid arguments (e.g. wrong variable type) are given!

Exercises

- Primer: “ATGCCATGCATTCGACTACG”
- 3.8) Calculate length of primer and print it!
- 3.9) Get number of Gs and print it!
- 3.10) Write a function to analyze the nucleotide composition of a primer and print it!
- 3.11) Is it a suitable primer? Why (not)?

Control structures

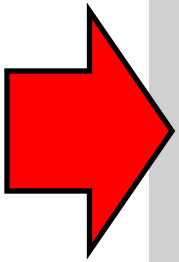
if & else

- Distinguish to cases:

```
1  a = 5   #define variable
2  #user inputs number:
3  b = input("please enter number!")
4  if b < a:   #if b is smaller than a
5      print "b is smaler than a"
6  else:   #in all other cases
7      print "b is NOT smaller than a"
```

elif

- Distinguish between multiple cases:



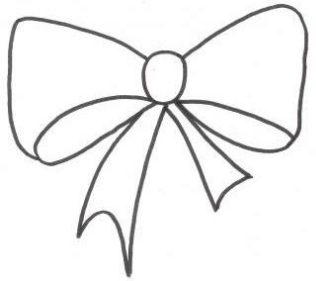
```
1  a = 5   #define variable
2  #user inputs number:
3  b = input("please enter number!")
4  if b < a:   #if b is smaller than a
5      print "b is smaler than a"
6  elif b == a:    #both are equal
7      print "b is matching a"
8  else:   #in all other cases
9      print "b is NOT smaller than a"
```

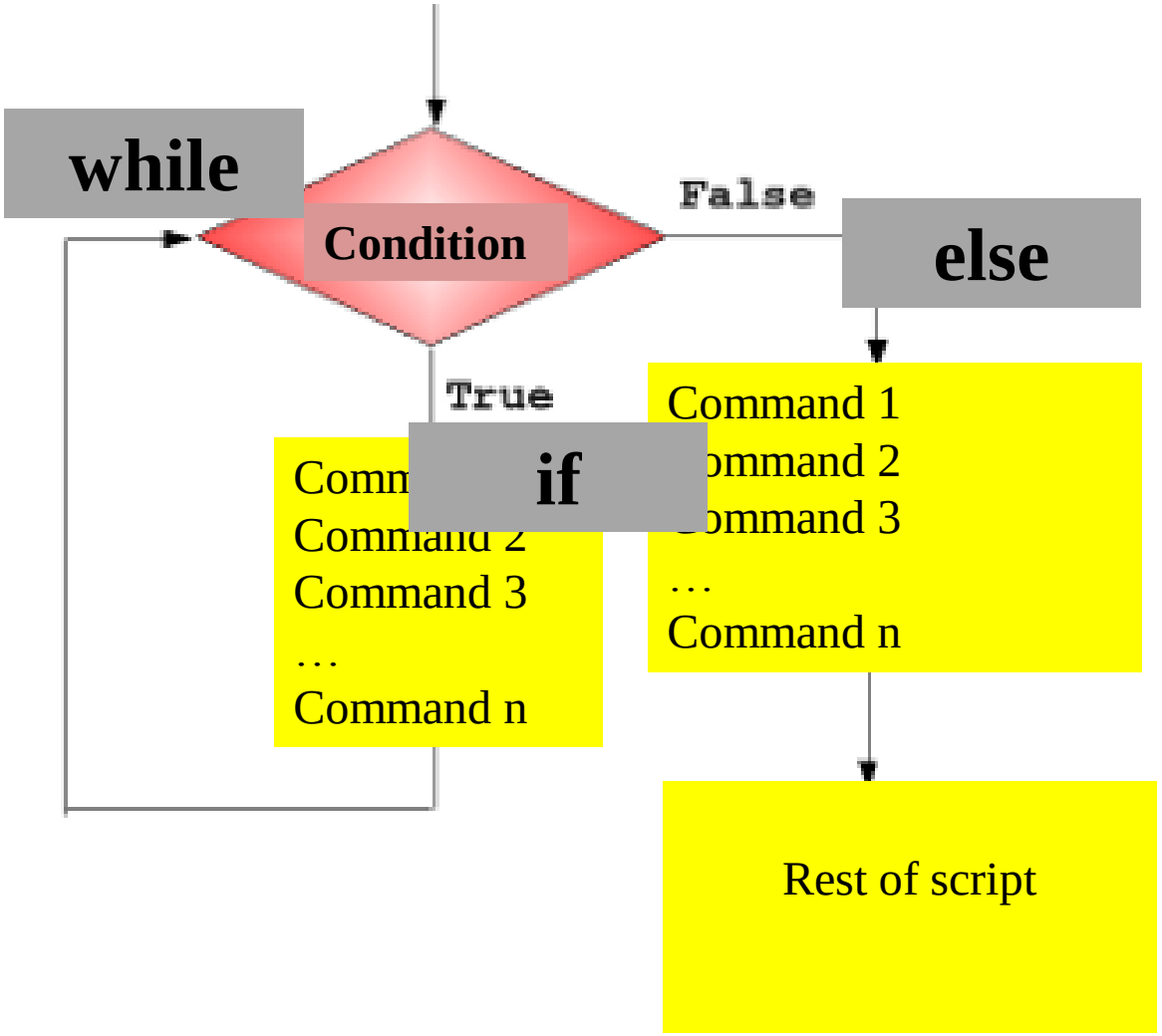
- Action depends on result of comparison

Exercises

- 4.1) Write script for guessing numbers!
- 4.2) Check if input is a number!
- 4.3) Check if input is float/integer!
- 4.4) Return error messages if wrong input is provided!
- 4.5) Add tips during the guessing process!

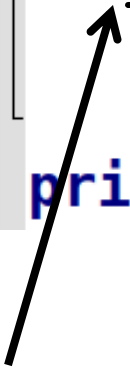
Concept of loops





While – loop (example)

```
1  a = 0
2  while a < 10:   #checks if a is smaller than 10
3      print str( a ) + " is smaller than 10"
4      a += 1  #a = a+1
5      #something useful could happen here
6      print "a was increased by 1"
7  print str( a ) + "is larger than 10"
```



Code is executed until the condition for this loop becomes false

While – infinite loop

```
1  #infinite loop:
2  a = 0
3  while True: #always true
4      a += 1  #a = a+1
5      print str( a )
6  print "this line is never reached"
```

WARNING: this loop is infinite!

For loop

```
1  list_of_species = [ "E.coli", "B.subtilis", "S.cerevisiae", "C.glutamicum", "A.tumefaciens" ]
2  for species in list_of_species:
3      if len( species ) < 12:  #Length of names is calculated and compared
4          print species        #Name is printed
5
6  #Line 3+4 is executed several times:
7  #1: species = "E.coli"
8  #2: species = "B.subtilis"
9  #3: species = "S.cerevisiae"
10 # ...
```

For loop

Control variable (species)

List of data (list_of_species)

```
1  list_of_species = [ "E.coli", "B.subtilis", "S.cerevisiae", "C.glutamicum", "A.tumefaciens" ]
2  for species in list_of_species:
3      if len( species ) < 12:  #Length of names is calculated and compared
4          print species        #Name is printed
5
6  #Line 3+4 is executed several times:
7  #1: species = "E.coli"
8  #2: species = "B.subtilis"
9  #3: species = "S.cerevisiae"
10 # ...
```

Species name is printed if shorter than 12 characters

Exercises

- 4.6) Write a function counting to 100 and printing all number which can be divided by 4 without any residue!
  - Info: 10 % 2 #modulo division in Python
- 4.7) Write a function counting down from 1000 to 0 and printing all numbers!
- 4.8) Generate a list of species names! Write a function printing all species names starting with “E”!
- 4.9) Expand this function to limit the printing to species names which are additionally shorter than 10 characters!
- 4.10) Expand this function to limit the printing to species names which are additionally ending with “a”.

range()

```
1  list_of_species = ["E.coli", "B.subtilis", "S.cerevisiae", "C.glutamicum", "A.tumefaciens"]
2  length = len( list_of_species ) #length = 5
3  for i in range( length ):   #starts at 0 and runs to i=4 (five values)
4      if len( list_of_species[ i ] ) < 12:    #length of name is calculated and compared
5          print list_of_species[ i ]  #name is printed
6
7  #i is taking five different values:
8  #1: i=0
9  #2: i=1
10 #3: i=2
11 #4: i=3
12 #5: i=4
13 #i=5 is never reached by range()
```

enumerate()

```
1  list_of_species = ["E.coli", "B.subtilis", "S.cerevisiae", "C.glutamicum", "A.tumefaciens"]
2  for idx, species in enumerate( list_of_species ):
3      if len( species ) < 12: #length of names is calculated and compared
4          print "position of " + species + " is: " + str( idx )
5
6  #i is taking five different values:
7  #1: i=0 and species="E.coli"
8  #2: i=1 and species="B.subtilis"
9  #3: i=2 and species="S.cerevisiae"
10 #4: i=3 and species="C.glutamicum"
11 #5: i=4 and species="A.tumefaciens"
12 #i=5 is never reached
```

Exercises

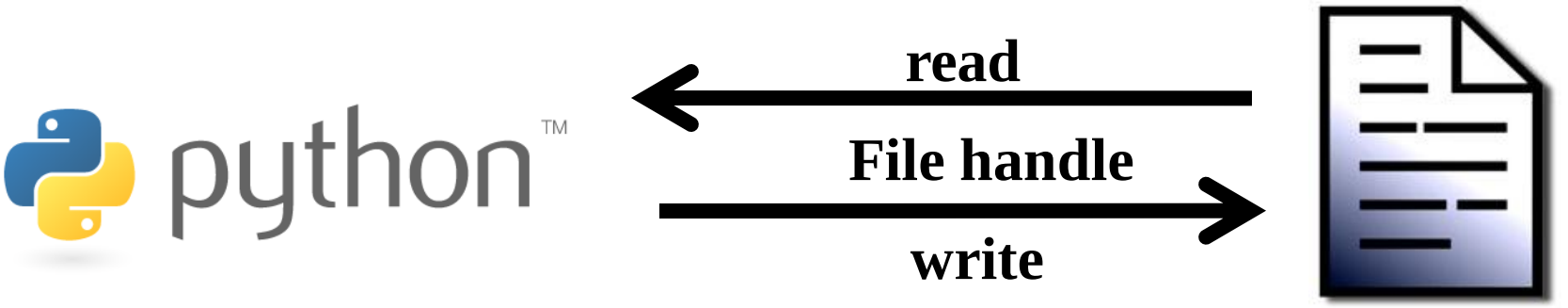
- 5.4) Write a script to print 50x “here” and the current value of the control variable!

- 5.5) Write a script to walk through the species list and to print the character from the species where the index corresponds to the current control variable value!

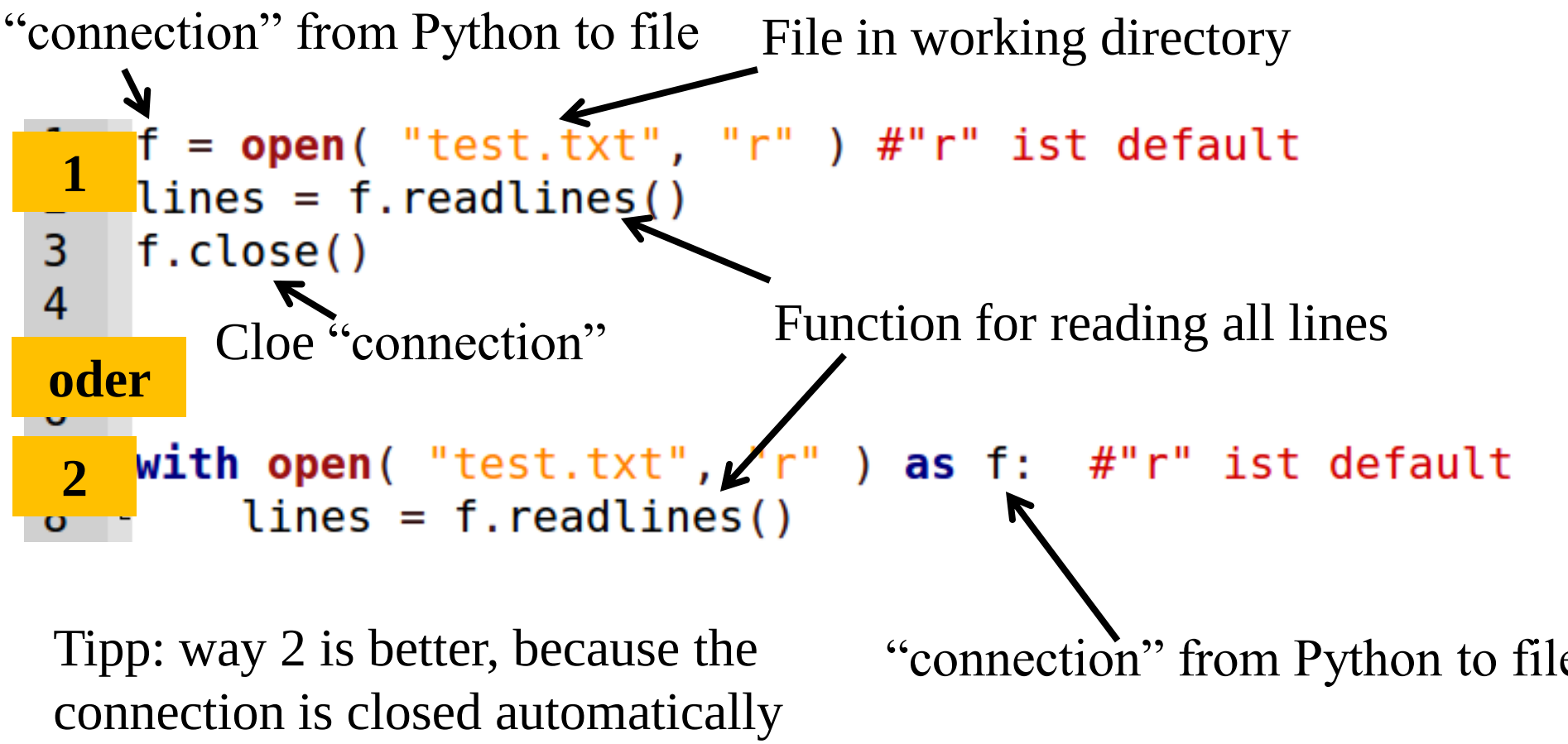
File handling

Concept of file handling

- “connection” from Python to file
- Read = Transfer of data **from** file
- Write = Transfer of data **into** file



Read a file (parsing)



Reading a file (big data)

```
1 with open( "test.txt", "r" ) as f:  #"r" ist default
2     line = f.readline() #liest die nächste (erste) Zeile aus Datei
3     while line:
4         print line
5         line = f.readline() #liest die nächste (erste) Zeile aus Datei
```

Until end of file is reached

- Advantage: only one line is read and processed at a time
- NGS data (e.g. FASTQ/SAM/BAM/VCF) are usually several GB in size => RAM limitations
- Very long sequence (e.g. genome sequences) in FASTA might be to large for available RAM

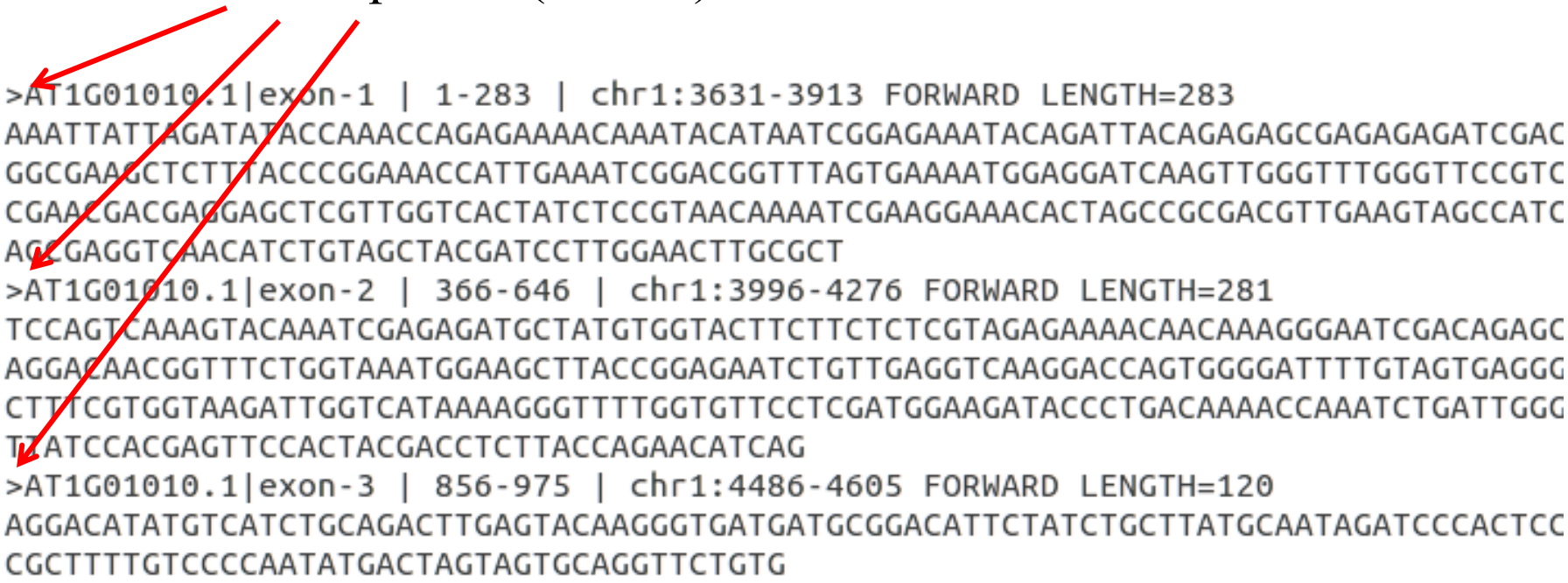
Analyze file - example

- How many lines are in AtCol0_Exons.fasta? (large file!)
- Under UNIX: head <DATEINAME>

```
>AT1G01010.1|exon-1 | 1-283 | chr1:3631-3913 FORWARD LENGTH=283
AAATTATTAGATATACCAAACCAGAGAAAACAAATACATAATCGGAGAAATACAGATTACAGAGAGCGAGAGAGATCGAC
GGCGAAGCTCTTTACCCGGAAACCATTGAAATCGGACGGTTTAGTGAAAATGGAGGATCAAGTTGGGTTTGGGTTCCGTC
CGAACGACGAGGAGCTCGTTGGTCACTATCTCCGTAACAAAATCGAAGGAAACACTAGCCGCGACGTTGAAGTAGCCATC
AGCGAGGTCAACATCTGTAGCTACGATCCTTGGAACTTGCGCT
>AT1G01010.1|exon-2 | 366-646 | chr1:3996-4276 FORWARD LENGTH=281
TCCAGTCAAAGTACAAATCGAGAGATGCTATGTGGTACTTCTTCTCTCGTAGAGAAAACAACAAAGGGAATCGACAGAGC
AGGACAACGGTTTCTGGTAAATGGAAGCTTACCGGAGAATCTGTTGAGGTCAAGGACCAGTGGGGATTTTGTAGTGAGGG
CTTTCGTGGTAAGATTGGTCATAAAAGGGTTTTGGTGTTCCTCGATGGAAGATACCCTGACAAAACCAAATCTGATTGGG
TTATCCACGAGTTCCACTACGACCTCTTACCAGAACATCAG
>AT1G01010.1|exon-3 | 856-975 | chr1:4486-4605 FORWARD LENGTH=120
AGGACATATGTCATCTGCAGACTTGAGTACAAGGGTGATGATGCGGACATTCTATCTGCTTATGCAATAGATCCCACTCC
CGCTTTTGTCCCCAATATGACTAGTAGTGCAGGTTCTGTG
```

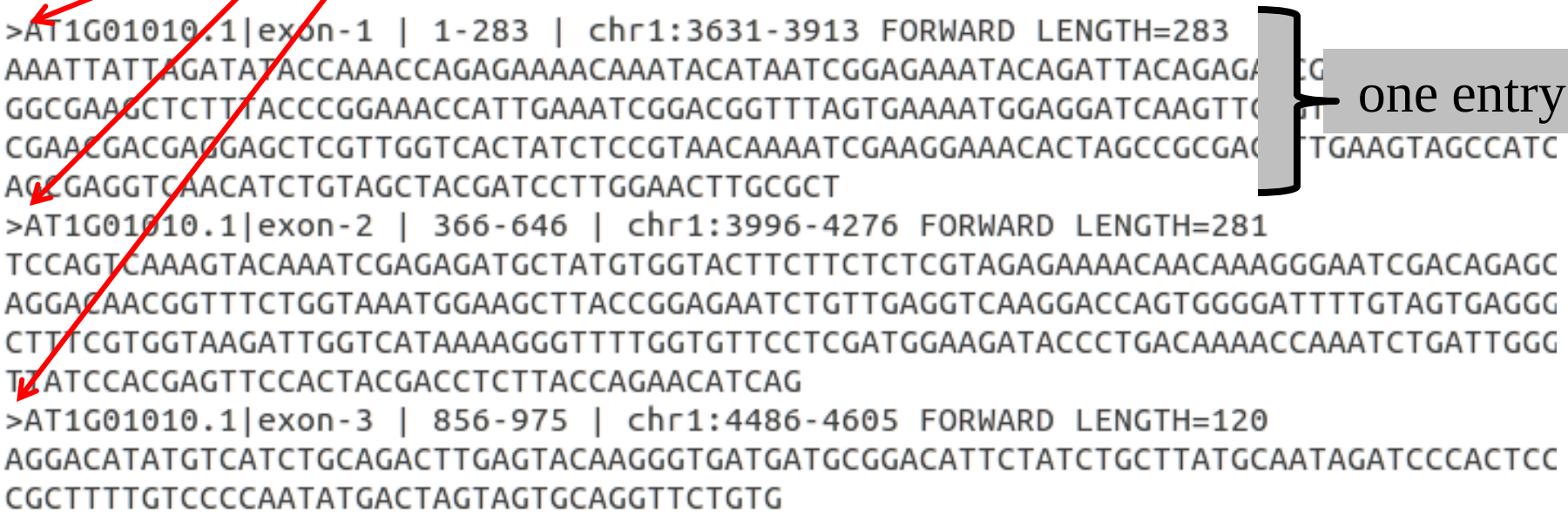
(multiple) FASTA

Name of sequence (header): line starts with '>'



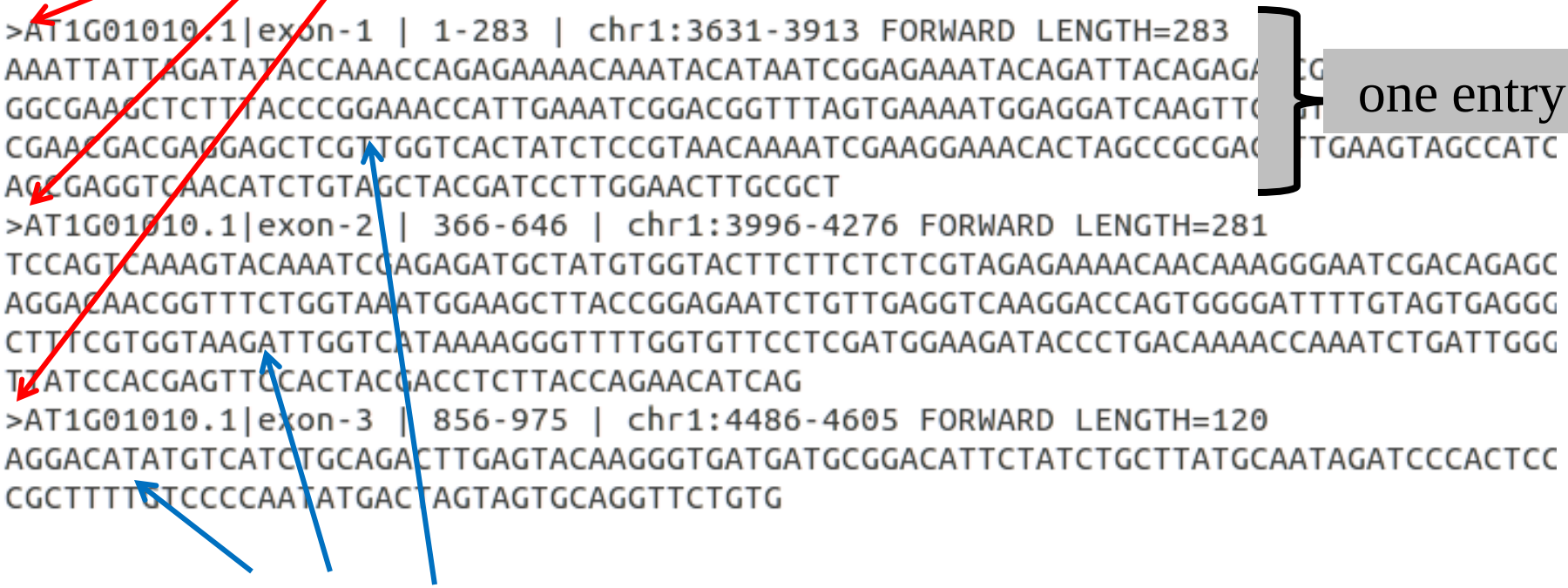
(multiple) FASTA

Name of sequence (header): line starts with '>'



(multiple) FASTA

Name of sequence (header): line starts with '>'



Analyze file - example

```
1  with open( "/vol/apbiokurs/data/AtCol0_Exons.fasta", "r" ) as f:
2      line = f.readline()
3      line_counter = 0
4      while line:
5          line_counter += 1
6          line = f.readline()
7
8  print  "File contains"  + str( line_counter ) +  " lines"
```

#First line

Counting lines

Converting number of lines in string

Exercises

- 6.1) Count number of sequences (= number of headers) in /vol/apbiokurs/data/AtCol0_Exons.fasta!
- 6.2) Count number of sequence lines!
- 6.3) Count number of characters in document! (How many per line?)
- 6.4) How long are all contained sequences combined?
- 6.5) Calculate the average sequence length in this file!

And back again… writing into file!

Read:

```
1
2  with open( "test.txt", "r" ) as f:  #"r" (read) ist default
3      lines = f.readlines()
4
5
6
7
8  Write:
9  with open( "test2.txt", "w" ) as out:
10     out.write( "hello world!" )
```

difference: r = read; w = write

Writes a string into a file

- If output file does not exists, it will be created!
- File handle (f and out) can have any name!

Read & write

```
1 with open( "test.txt", "r" ) as f:  #open file to read
2     with open( "test2.txt", "w" ) as out:   #open file to write
3         line = f.readline() #read from first file
4         while line:
5             #this would be the place to apply filters
6             out.write( line )
7             line = f.readline()
```

Exercises

- 7.1) Read the file AtCol0_Exons.fasta and write all headers (starting with '>') into a new file!

- 7.2) Read the file AtCol0_Exons.fasta and write the following:
  - Line if it is a header
  - Length of line if it is a sequence line

- 7.3) Calculate the number of sequences, the cumulative length, and the average length in the new file! Are they matching the values of the original file?

- 7.4) Write sequences into a new file if there length is a multiple of 10!

White space characters

- New line (‘\n’) und tab (‘\t’) are special characters

  print “hello\tworld!\nhello\tworld!\n”

- Python interprets these characters in print statements, but functions like readline() and write() do not!

  => New line needs to be added “manually” to each new line

```
1 with open( "test3", "w" ) as out:
2     out.write( "erster test" )
3     out.write( "zweiter test" )
4     out.write( "dritter test\n" )
5     out.write( "vierter test\n" )
6     out.write( "fünfter test" )
7     out.write( "sechster test" )
```

How many lines does this file have?

strip()

- Removes white space characters from borders of a string (often used for new lines at the line end):

```
1  line = ">name_of_first_seq\n"
2  print line
3  #>name_of_first_seq
4  # [empy line genereated by \n ]
5  line = line.strip()
6  print line
7  #>name_of_first_seq
```

split()

- Separates a string at each given occurrence of the given substring (e.g. tab, comma, …)

- Generates list of strings

```
1  #tab-delimited file
2  line = "spalte1\tspalte2\tspalte3\tspalte4\n"
3  #line should be splitted at tabs
4  columns = line.strip().split('\t')
5  print columns
6  #["spalte1", "spalte2", "spalte3", "spalte4"]
```

join()

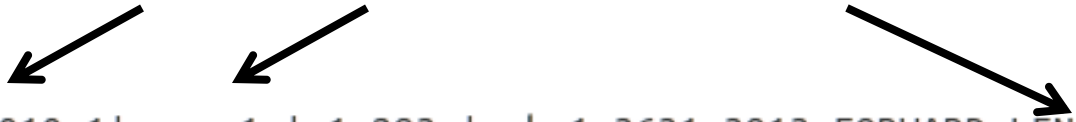
- Combines strings of a list by putting a given substring between them (e.g. underline)

- Important: all elements of list need to be strings!

```
1  #tab-delimited file
2  line = "spalte1\tspalte2\tspalte3\tspalte4\n"
3  #line should be splitted at tabs
4  columns = line.strip().split('\t')
5  print columns
6  #["spalte1", "spalte2", "spalte3", "spalte4"]
7
8  new_line = "_".join( columns )
9  print new_line
10 #spalte1_spalte2_spalte3_spalte4
```

Exercises

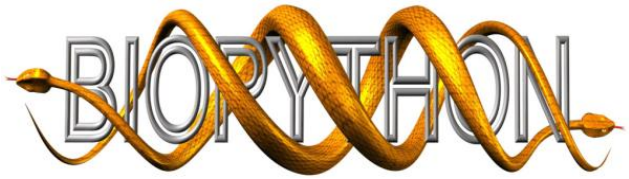
- 7.5) Read the file AtCol0_Exons.fasta and write the following:
  - Only ArabidopsisGeneIdentifier (e.g. AT1G01010)
  - Gene identifier, exon name, and exon length (tab-delimited)



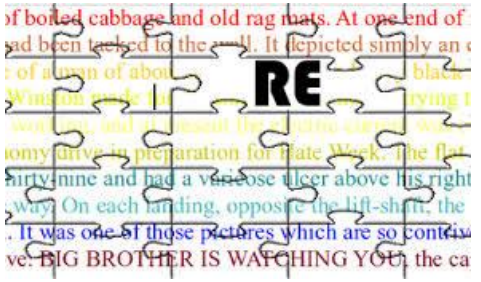
```
>AT1G01010.1|exon-1 | 1-283 | chr1:3631-3913 FORWARD LENGTH=283
AAATTATTAGATATACCAAACCAGAGAAAACAAATACATAATCGGAGAAATACAGATTACAGAGAGCGAGAGAGATCGAC
GGCGAAGCTCTTTACCCGGAAACCATTGAAATCGGACGGTTTAGTGAAAATGGAGGATCAAGTTGGGTTTGGGTTCCGTC
CGAACGACGAGGAGCTCGTTGGTCACTATCTCCGTAACAAAATCGAAGGAAACACTAGCCGCGACGTTGAAGTAGCCATC
AGCGAGGTCAACATCTGTAGCTACGATCCTTGGAACTTGCGCT
>AT1G01010.1|exon-2 | 366-646 | chr1:3996-4276 FORWARD LENGTH=281
TCCAGTCAAAGTACAAATCGAGAGATGCTATGTGGTACTTCTTCTCTCGTAGAGAAAACAACAAAGGGAATCGACAGAGC
AGGACAACGGTTTCTGGTAAATGGAAGCTTACCGGAGAATCTGTTGAGGTCAAGGACCAGTGGGGATTTTGTAGTGAGGG
CTTTCGTGGTAAGATTGGTCATAAAAGGGTTTTGGTGTTCCTCGATGGAAGATACCCTGACAAAACCAAATCTGATTGGG
TTATCCACGAGTTCCACTACGACCTCTTACCAGAACATCAG
>AT1G01010.1|exon-3 | 856-975 | chr1:4486-4605 FORWARD LENGTH=120
AGGACATATGTCATCTGCAGACTTGAGTACAAGGGTGATGATGCGGACATTCTATCTGCTTATGCAATAGATCCCACTCC
CGCTTTTGTCCCCAATATGACTAGTAGTGCAGGTTCTGTG
```

Module

Concept of modules



NcbiblastnCommandline
NCBIXML



re.findall()
re.match()

plt.show()
plt.color()

np.array()

Import of modules

```
 1  #basic import:
 2  import re
 3  #import of module under abbreviation:
 4  import numpy as np
 5  #import part of module:
 6  from datetime import datetime
 7
 8  #usage of module functions:
 9  re.findall()
10  datetime.now()
11
12  #try this (requires import):
13  print str( datetime.now() )
```

Run time calculation

- Current time is saved in two different places
- Difference is calculated to get the run time

```
1  from datetime import datetime
2
3  t1 = datetime.now()
4  #something should happen here
5  t2 = datetime.now()
6
7  print "it took " + str( t2-t1 )
```

Regular expressions

- regular expressions (= re) enable efficient search for substrings in a given string

```
1  import re
2  some_string = "AT2G12340.1|exon-1|23745-23965|AT2G12340.2exon-1_23745-23965"
3  hits = re.findall( "AT\dG\d{5}", some_string )  #generates list of hits
4  #searches for "AT\dG\d{5}"
5  #AT, G are matching the very same character
6  #\d is matching all number 0-9
7  #{5} specifies five repetitions of the previous element
8
9  print hits
```

Exercises

- 8.1) Write all AGIs of AtCol0_exons.fasta into a new file!
- 8.2) Some IDs occur multiple times. Add a filter step to reduce the results to unique IDs!
- 8.3) Calculate frequency of each AGI and construct a histogram (matplotlib)!

DNA-, RNA- and AA sequences

Reverse complement

What happens here?

```
1  def revcomp( seq ):
2
3      seq = seq.lower()
4
5      #key:value (=dictionary)
6      complement = { 'a':'t', 't':'a', 'c':'g', 'g':'c' }
7
8      new_seq = []
9
10     for nt in seq:
11         new_seq.append( complement[ nt ] )
12
13     #list[::-1] inverts list (last element becomes first)
14     new_seq = "".join( new_seq[::-1] )
15
16     return new_seq
```

Sequence of bases e.g. ATGACATGA

Converts input to lower case: atgacatga

Get complement for each base

Inverts list (=reverse)

Exercises

- 9.1) Write a function to get the reverse complement (upper case letters) of a DNA sequence given in upper case letters!

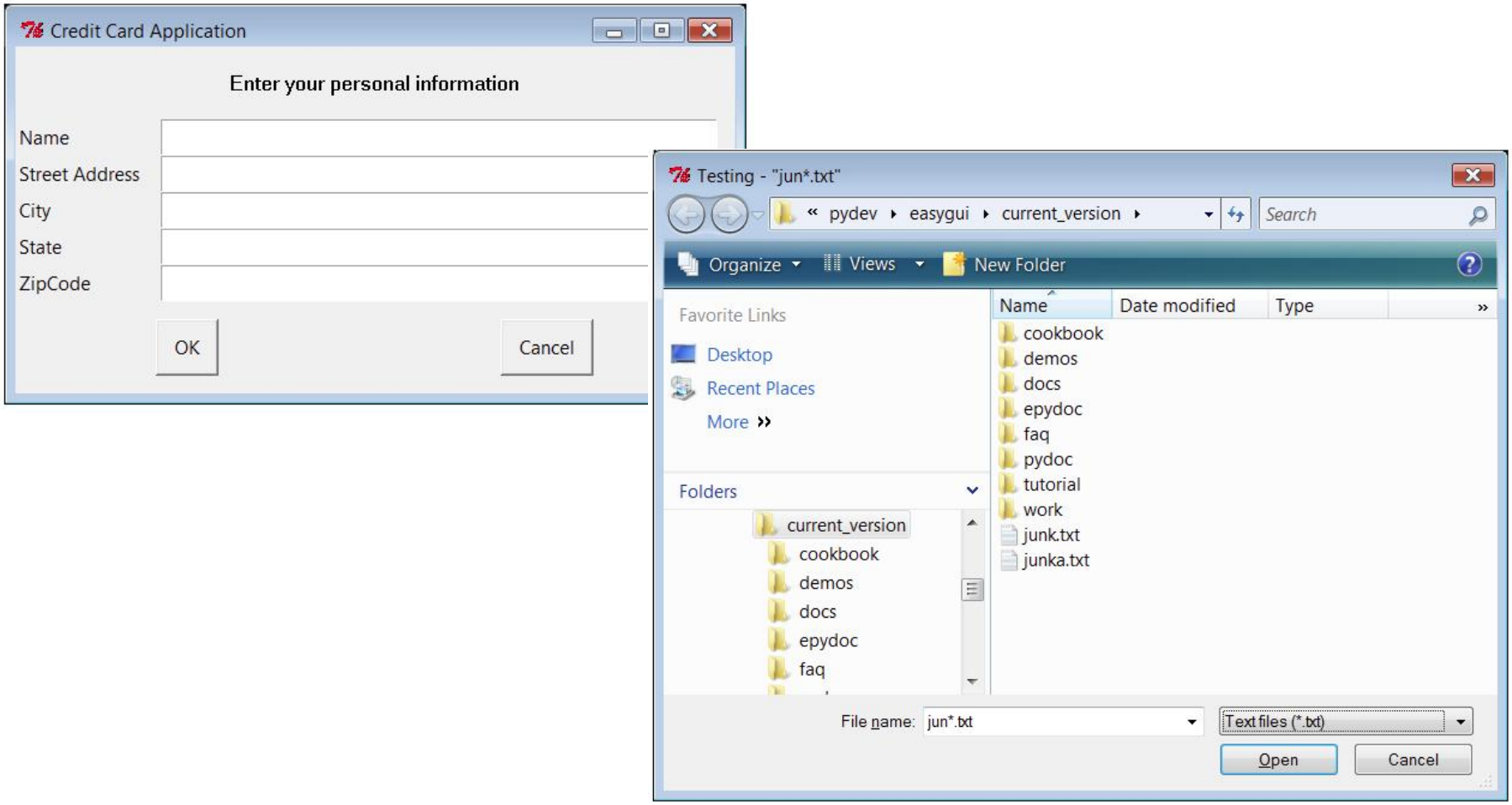
- 9.2) Write a function to convert a DNA sequence into a RNA sequence!

- 9.3) Write a function to translate a DNA sequence into amino acids (first frame only)!

  Tipp: http://en.wikipedia.org/wiki/DNA_codon_table

- 9.4) Write a function to translate DNA sequences in all 6 frames into peptide sequences! The longest peptide sequence per DNA sequence should be returned!

EasyGUI


Credit Card Application
Enter your personal information
Name
Street Address
City
State
ZipCode
OK
Cancel
Testing - "jun*.txt"
pydev ▸ easygui ▸ current_version ▸
Search
Organize
Views
New Folder
Favorite Links
Desktop
Recent Places
More ››
Folders
current_version
cookbook
demos
docs
epydoc
faq
Name
Date modified
Type
cookbook
demos
docs
epydoc
faq
pydoc
tutorial
work
junk.txt
junka.txt
File name: jun*.txt
Text files (*.txt)
Open
Cancel

EasyGUI - documentation

http://www.ferg.org/easygui/tutorial.html

http://easygui.sourceforge.net/tutorial.html#introduction

https://easygui.wordpress.com/

Downloads:
https://pypi.python.org/pypi/easygui/0.97.4#downloads

Two issues

Easygui project shuts down

Posted on 2013/03/06

Effective March 6, 2013, I am shutting down the EasyGui project.

The EasyGui software will continue to be available at its current location, but I will no longer be supporting, maintaining, or enhancing it.

The reasons for this decision are personal, and not very interesting. I'm older now, and retired. I no longer do software development, in any programming language. I have other interests that I find more compelling. I spend time with my family. I play and promote petanque. Life is good, but it is different.

During the course of my software development career I've had occasion to shut down a number of projects. On every occasion when I turned over a project to a new owner, the results were disappointing. Consequently, I have decided to shut down the EasyGui project rather than to try to find a new owner for it.

The EasyGui software will remain frozen in its current state. I invite anyone who has the wish, the will, the energy, and the vision to continue to evolve EasyGui, to do so. Copy it, fork it, and make it the basis for your own new work.

— Steve Ferg, March 6, 2013

Warning about using EasyGui with IDLE

… but it should work!

Selected EasyGUI functions

- msgbox = disaplays message box
- ccbox / ynbox = poll with yes/no box
- buttonbox = diverse buttons displayed (suitable as menu)
- choicebox = select options from a given list
- enterbox = enter some text
- diropenbox/fileopenbox/filesavebox = GUI for file handling

Basic structure

```
 1  import easygui as eg
 2  import sys
 3
 4  while 1:
 5      #
 6      #your functions / boxes
 7      #
 8      msg = "do you want to continue?"
 9      title = "tool usage"
10      if eg.ccbox( msg, title ):
11          pass
12      else:
13          sys.exit( 0 )
```

Exercises

- 10.1) Write a script to handle input of primer names and sequences! All information should be saved in a multiple FASTA file.

- 10.2) Write a script to return a matching primer sequence from a FASTA file based on a given primer name.

- 10.3) Write a script to combine both functionalities: return primer sequence, if name is already present OR generate new entry if primer name is novel.

Advanced Python

Boas Pucker

Overview

- Submission of processes to shell for execution and result handling
- BLAST result analysis
- Concept of plots via axes in matplotlib
- General figure types: plot, boxplot, barplot
  - Genome figures: scatter plots, box plots, chromosome plots, gene cluster plots
- Statistics in Python: tests and theoretical background
  - t-test, W-test, U-test, cor-test, X^2-test
- HTML construction

Analyze problem

- Split problem into parts => solutions for small parts are more generic and thus often available

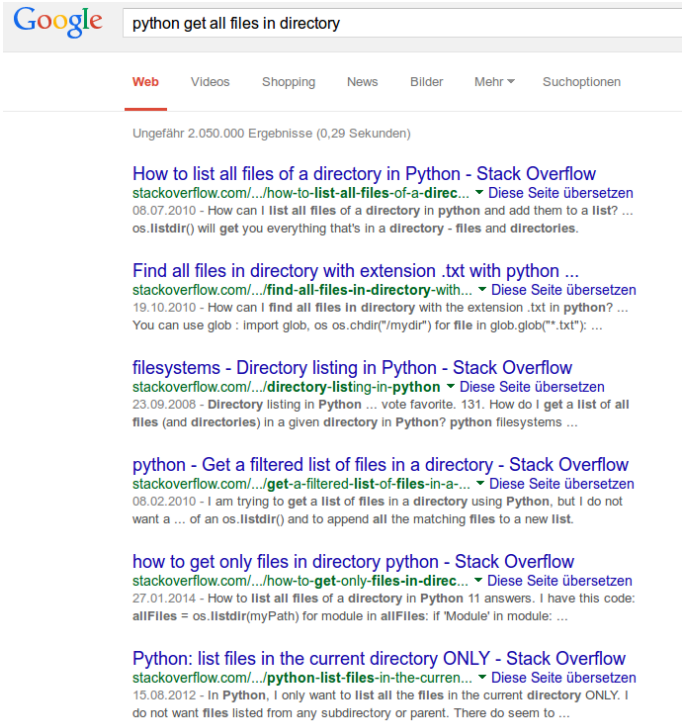
- Precise description of problem is necessary:

stackoverflow

- Best hit in approximately 90-95% of all cases
- Stackoverflow is chat forum for programmers (different languages)
- Problems are described in questions
- Answers often contain examples which can be directly applied
- Answers are judged by members => look for green marking and the number of positive votes

Example

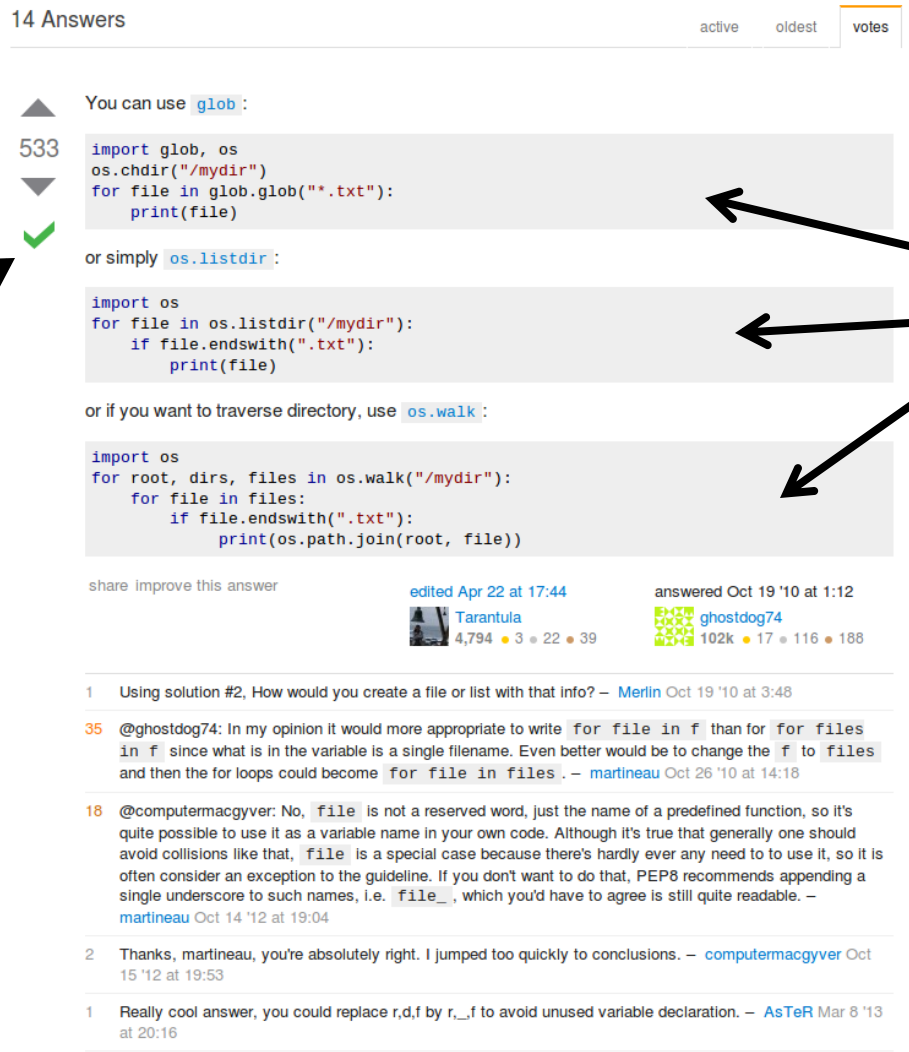
- Problem: get paths of all files in a certain directory
- Search expression: “python get all files in directory”



Example

533 like this answer

This answers solved the question



14 Answers

active oldest votes

533

You can use `glob`:

```
import glob, os
os.chdir("/mydir")
for file in glob.glob("*.txt"):
    print(file)
```

or simply `os.listdir`:

```
import os
for file in os.listdir("/mydir"):
    if file.endswith(".txt"):
        print(file)
```

or if you want to traverse directory, use `os.walk`:

```
import os
for root, dirs, files in os.walk("/mydir"):
    for file in files:
        if file.endswith(".txt"):
            print(os.path.join(root, file))
```

share improve this answer

edited Apr 22 at 17:44 Tarantula 4,794 • 3 • 22 • 39

answered Oct 19 '10 at 1:12 ghostdog74 102k • 17 • 116 • 188

1 Using solution #2, How would you create a file or list with that info? – Merlin Oct 19 '10 at 3:48

35 @ghostdog74: In my opinion it would more appropriate to write `for file in f` than for `for files in f` since what is in the variable is a single filename. Even better would be to change the `f` to `files` and then the for loops could become `for file in files`. – martineau Oct 26 '10 at 14:18

18 @computermacgyver: No, `file` is not a reserved word, just the name of a predefined function, so it's quite possible to use it as a variable name in your own code. Although it's true that generally one should avoid collisions like that, `file` is a special case because there's hardly ever any need to to use it, so it is often consider an exception to the guideline. If you don't want to do that, PEP8 recommends appending a single underscore to such names, i.e. `file_`, which you'd have to agree is still quite readable. – martineau Oct 14 '12 at 19:04

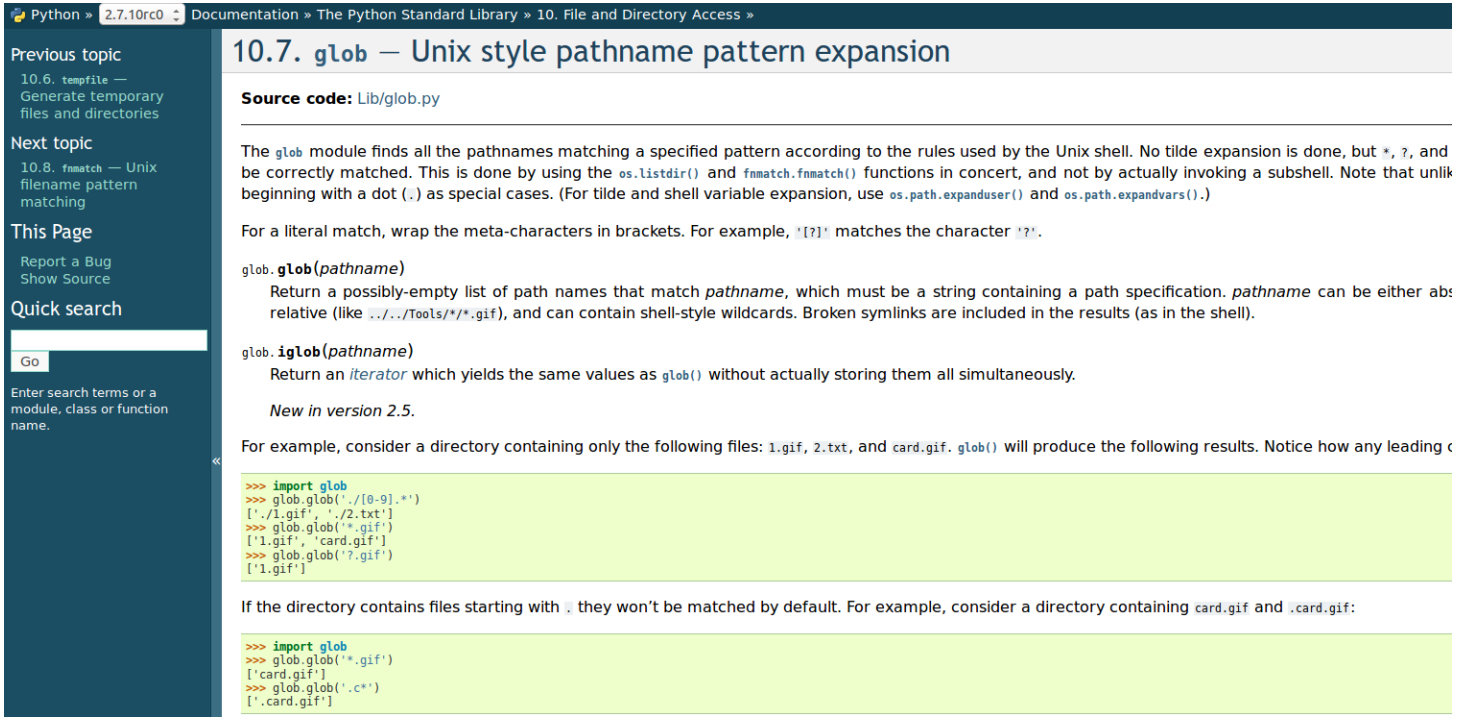
2 Thanks, martineau, you're absolutely right. I jumped too quickly to conclusions. – computermacgyver Oct 15 '12 at 19:53

1 Really cool answer, you could replace r,d,f by r,_,f to avoid unused variable declaration. – AsTeR Mar 8 '13 at 20:16

Three different ways to solve this problem are described (my favorite is not included)

Official Python documentation

- Systematic documentation of all functions in a module with all possible arguments
- Sometimes examples are given as well



Python » 2.7.10rc0 Documentation » The Python Standard Library » 10. File and Directory Access »

Previous topic
10.6. tempfile — Generate temporary files and directories

Next topic
10.8. fnmatch — Unix filename pattern matching

This Page
Report a Bug
Show Source

Quick search
Go
Enter search terms or a module, class or function name.

10.7. glob — Unix style pathname pattern expansion

Source code: Lib/glob.py

The glob module finds all the pathnames matching a specified pattern according to the rules used by the Unix shell. No tilde expansion is done, but *, ?, and be correctly matched. This is done by using the os.listdir() and fnmatch.fnmatch() functions in concert, and not by actually invoking a subshell. Note that unlik beginning with a dot (.) as special cases. (For tilde and shell variable expansion, use os.path.expanduser() and os.path.expandvars().)

For a literal match, wrap the meta-characters in brackets. For example, '[?]' matches the character '?'.

glob.**glob**(*pathname*)
Return a possibly-empty list of path names that match *pathname*, which must be a string containing a path specification. *pathname* can be either abs relative (like ../../Tools/*/*.gif), and can contain shell-style wildcards. Broken symlinks are included in the results (as in the shell).

glob.**iglob**(*pathname*)
Return an *iterator* which yields the same values as glob() without actually storing them all simultaneously.

New in version 2.5.

For example, consider a directory containing only the following files: 1.gif, 2.txt, and card.gif. glob() will produce the following results. Notice how any leading c

```
>>> import glob
>>> glob.glob('./[0-9].*')
['./1.gif', './2.txt']
>>> glob.glob('*.gif')
['1.gif', 'card.gif']
>>> glob.glob('?.gif')
['1.gif']
```

If the directory contains files starting with . they won't be matched by default. For example, consider a directory containing card.gif and .card.gif:

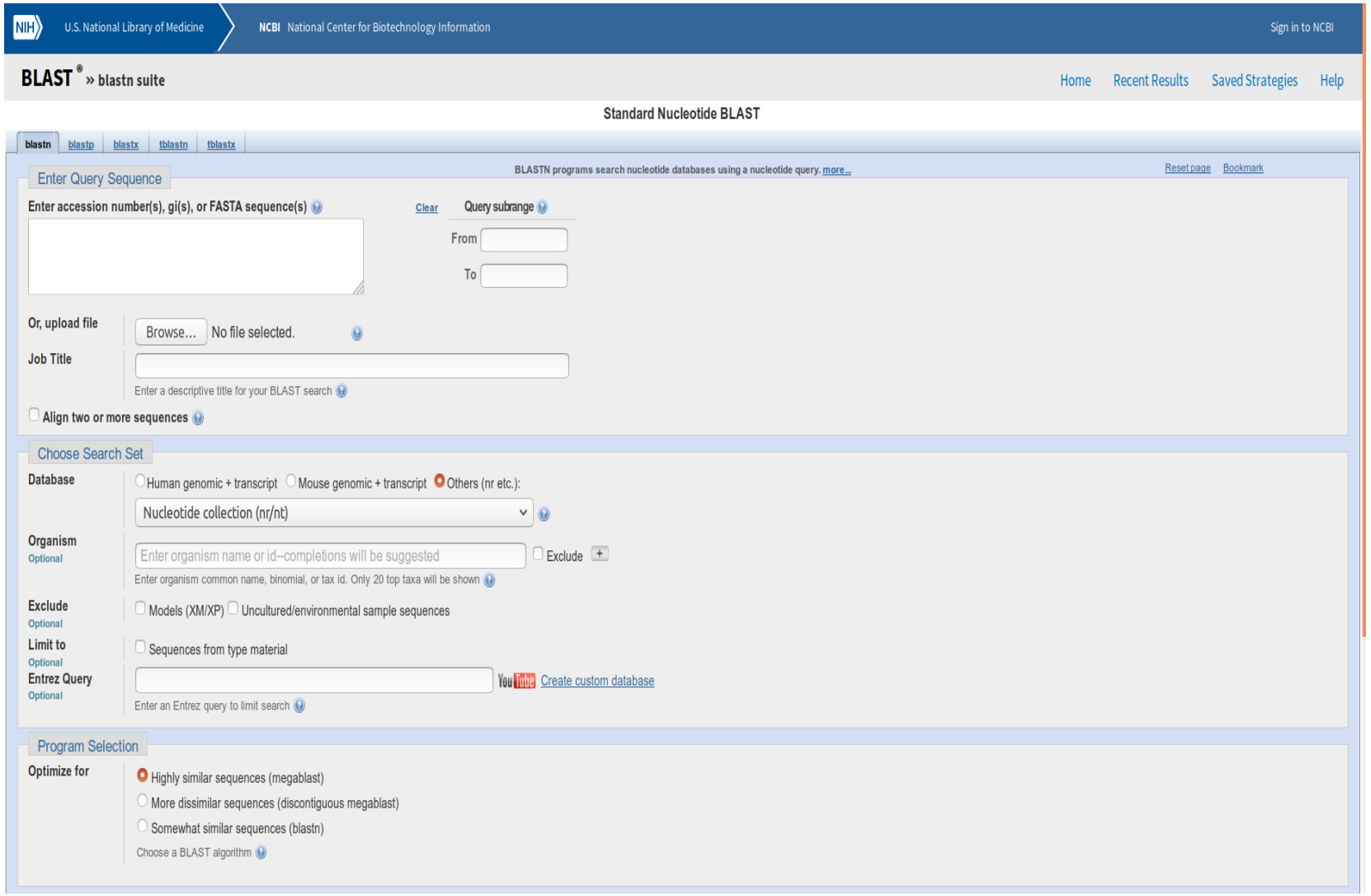
```
>>> import glob
>>> glob.glob('*.gif')
['card.gif']
>>> glob.glob('.c*')
['.card.gif']
```

Books?

- There is enough out there ….
  - Google's Python Class: https://developers.google.com/edu/python/
  - Youtube: https://www.youtube.com/watch?v=tKTZoB2Vjuk

- Book:
  - Well written with nice illustrations
  - Source of some presented figures

BLAST


U.S. National Library of Medicine
NCBI National Center for Biotechnology Information
Sign in to NCBI
BLAST® » blastn suite
Home
Recent Results
Saved Strategies
Help
Standard Nucleotide BLAST
blastn
blastp
blastx
tblastn
tblastx
BLASTN programs search nucleotide databases using a nucleotide query. more...
Reset page
Bookmark
Enter Query Sequence
Enter accession number(s), gi(s), or FASTA sequence(s)
Clear
Query subrange
From
To
Or, upload file
Browse...
No file selected.
Job Title
Enter a descriptive title for your BLAST search
Align two or more sequences
Choose Search Set
Database
Human genomic + transcript
Mouse genomic + transcript
Others (nr etc.):
Nucleotide collection (nr/nt)
Organism
Optional
Enter organism name or id--completions will be suggested
Exclude
Enter organism common name, binomial, or tax id. Only 20 top taxa will be shown
Exclude
Optional
Models (XM/XP)
Uncultured/environmental sample sequences
Limit to
Optional
Sequences from type material
Entrez Query
Optional
Create custom database
Enter an Entrez query to limit search
Program Selection
Optimize for
Highly similar sequences (megablast)
More dissimilar sequences (discontiguous megablast)
Somewhat similar sequences (blastn)
Choose a BLAST algorithm

How to execute processes via shell

- os.popen(command)

- Can be used to do almost everything via Python
- Python stops at line until command is completed
- Example:

  os.popen(‘mkdir my_new_popen_test‘)

Running BLAST

- BLAST = Basic Local Alignment Search Tool

  blastn –query <query_file> \

  -subject <subject_file> \ #-db <db_name>

  -out <output_file> \

  -outfmt 6
- Useful arguments:

  -evalue <FLOAT> #e-value cutoff

  -max_target_seqs <INT> #number of target seqs to analyze

  -num_threads <INT> #number of parallel threads

BLAST results (outfmt 6)

1.	qseqid	query (e.g., gene) sequence id
2.	sseqid	subject (e.g., reference genome) sequence id
3.	pident	percentage of identical matches
4.	length	alignment length
5.	mismatch	number of mismatches
6.	gapopen	number of gap openings
7.	qstart	start of alignment in query
8.	qend	end of alignment in query
9.	sstart	start of alignment in subject
10.	send	end of alignment in subject
11.	evalue	expect value
12.	bitscore	bit score

(source: http://www.metagenomics.wiki/tools/blast/blastn-output-format-6)

BLAST result parsing

```
 1  def load_BLAST_results( input_file ):
 2      """! @brief load all BLAST results from file """
 3
 4      data = []
 5      with open( input_file, "r" ) as f:
 6          line = f.readline()
 7          while line:
 8              parts = line.strip().split('\t')
 9              data.append( {  'query': parts[0],
10                              'subject': parts[1],
11                              'query_start': int( parts[6] ),
12                              'query_end': int( parts[7] ),
13                              'score': float( parts[-1] )
14                          } )
15              line = f.readline()
16      return data
```

How to organize a Python script …

- Executable script under LINUX:

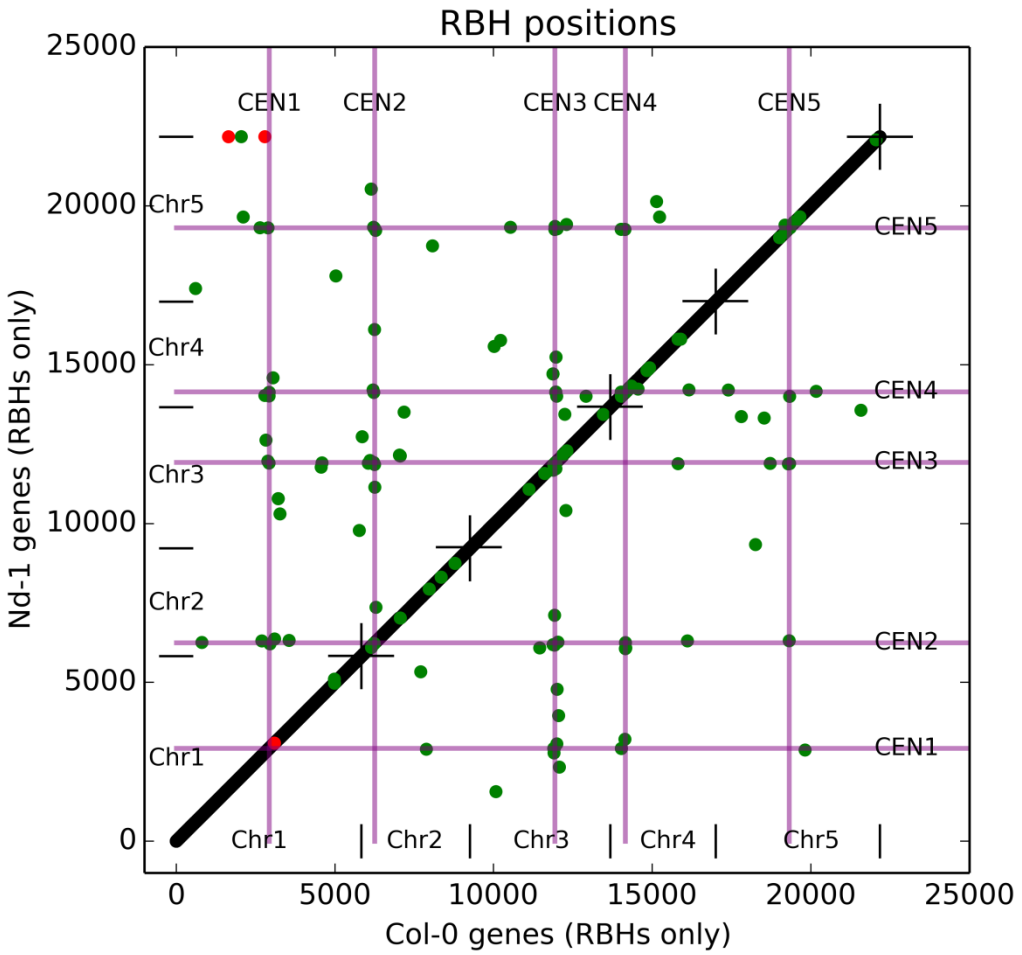
#!/usr/bin/env python #takes python version in path

#!/usr/bin/python #specifies a specific python version

- Author
- Version
- Imports
- Usage

matplotlib

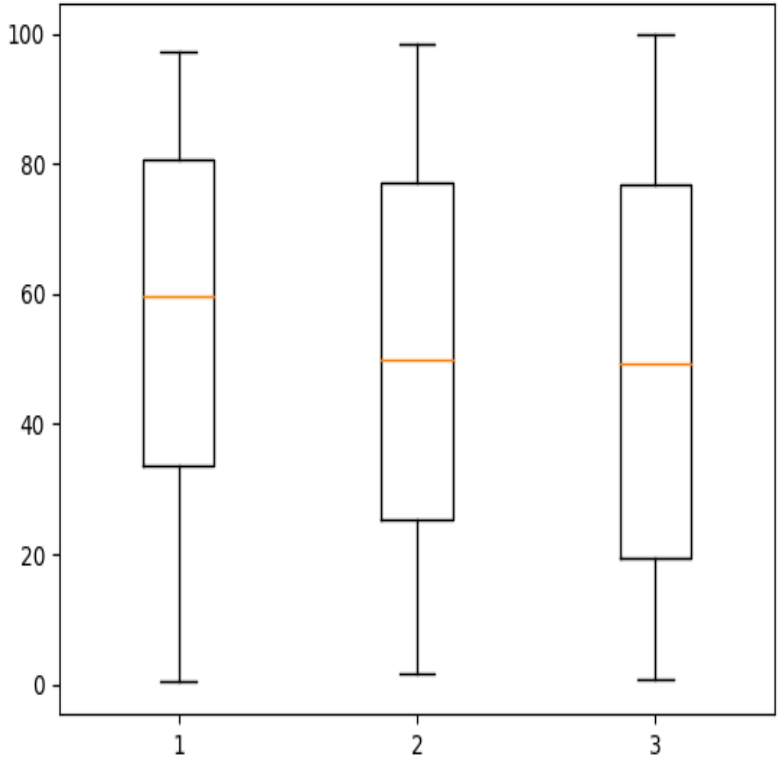
- „import matplotlib.pyplot as plt“
- Visualization of complex data
- Automatic generation of plots
- Amazing customization options



(Pucker et al., 2016)

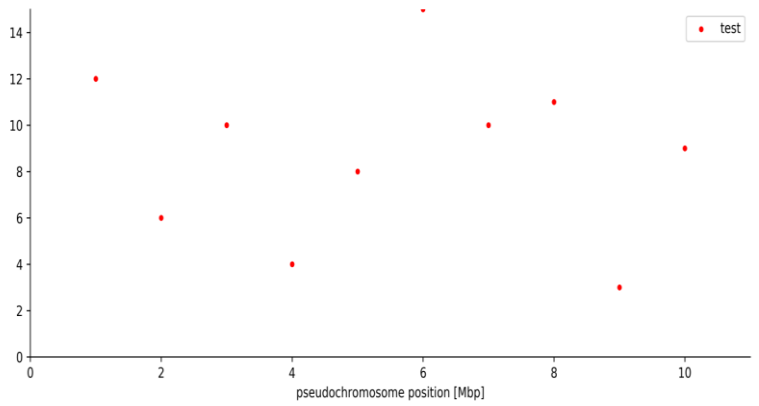
Box plot

```
1   import matplotlib.pyplot as plt
2   import numpy as np
3
4   d1 = np.random.rand(50) * 100   #generate random numbers
5   d2 = np.random.rand(50) * 100
6   d3 = np.random.rand(50) * 100
7
8   data = [d1, d2, d3] # multiple box plots on one figure
9
10  plt.figure()
11  plt.boxplot(data)
12  plt.show()
```



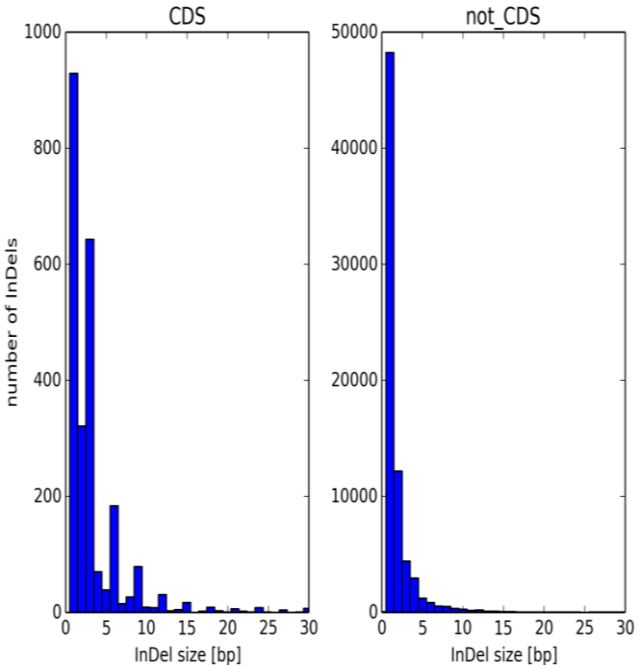
Scatter plot

```python
import matplotlib.pyplot as plt


fig, ax = plt.subplots( figsize=( 10, 4 ) ) #defining size of plot

x_values = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]
y_values = [ 12, 6, 10, 4, 8, 15, 10, 11, 3, 9 ]

ax.scatter( x_values, y_values, color="red", s=10, marker="o", label="test"
#setting color, marker size, marker shape and label of this group

ax.legend( numpoints=1 )
#each group is represented by only one marker in the legend (default=3)

ax.set_ylim( 0, 15 )    #set range of x-axis)
ax.set_xlim( 0, 11 )    #set range of y-axis
ax.set_xlabel( "pseudochromosome position [Mbp]" )

ax.spines["top"].set_visible(False)     #remove lines and ticks
ax.spines["right"].set_visible(False)   #remove lines and ticks

plt.subplots_adjust(left=0.05, right=0.99, top=0.97, bottom=0.12)
#adjust size of plot within figure

plt.show()
fig.savefig( "my_plot.png", dpi=600 )   #write figure into output file
plt.close( "all" )  #destroy created figures (cleaning up)
```



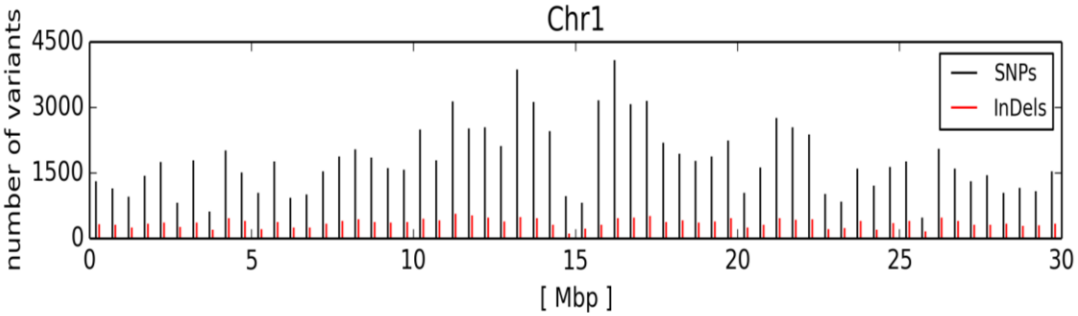
Histogram

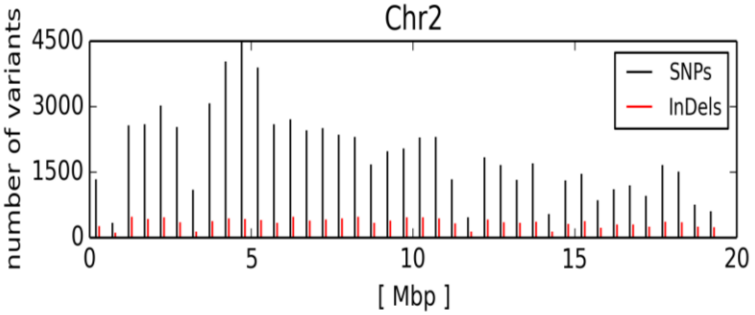
```
1  import matplotlib.pyplot as plt
2
3  # --- end of imports --- #
4
5  gene_space = [  3, 3, 6, 6, 9, 9, 12, 3, 3, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
6                  11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
7                  12, 15, 18, 21, 24, 27, 30 ]
8  intergenic = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5, 6, 7, 8, 9,
9                  1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 4, 1, 2, 1 ]
10
11
12 fig, ( ax1, ax2 ) = plt.subplots( 1, 2, sharey=False)
13 counts, bins, patches = ax1.hist( gene_space, bins=max( gene_space ), align="left" )
14 ax1.set_title( "CDS" )
15 ax1.set_xlim( 0, 30 )
16 ax1.set_xlabel( "InDel size [bp]" )
17 ax1.set_ylabel( "number of InDels" )
18
19 counts, bins, patches = ax2.hist( intergenic, bins=max( intergenic ), align="left" )
20 ax2.set_title( "not_CDS" )
21 ax2.set_xlim( 0, 30 )
22 ax2.set_xlabel( "InDel size [bp]" )
23 plt.subplots_adjust( wspace=0.3 )   #increase space between figures
24
25 plt.show()
26 fig.savefig( prefix + "InDel_size_distribution.png", dpi=300 )
27 plt.close('all')
```

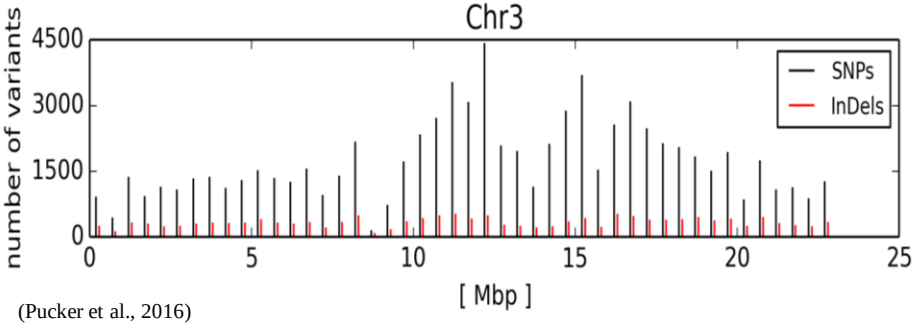


(Pucker et al., 2016)

‘barplot‘ figure



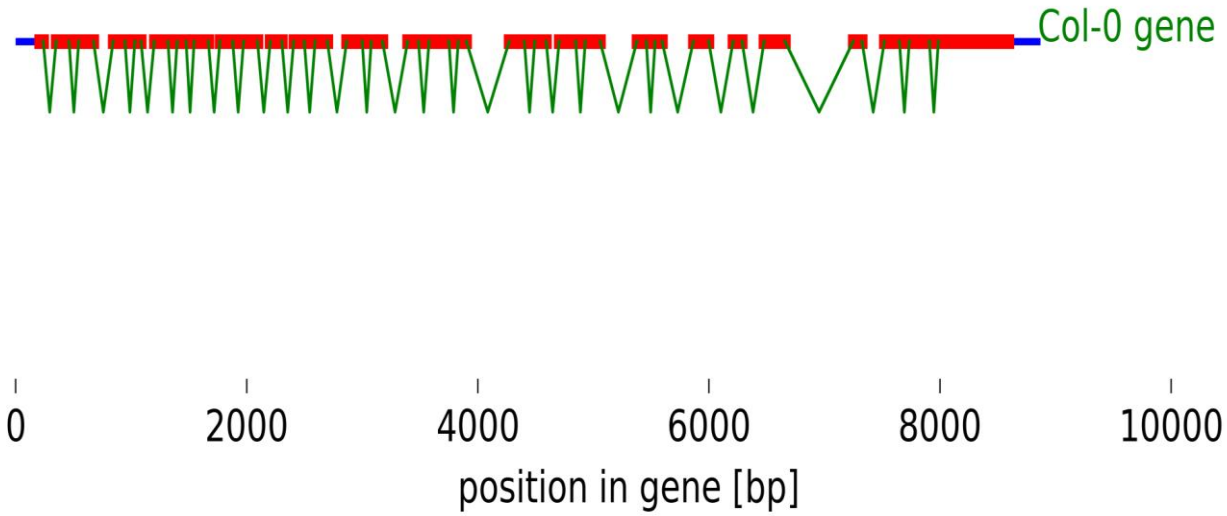




(Pucker et al., 2016)

barplots.py generates barplots at specific positions by drawing a normal line

Gene structure plot



(Pucker et al., 2017)

gene_structure_plot.py generates visualizations of gene/transcript structures based on GFF annotations

Exercise: operon structure plot

- Construct a figure to illustrate the order and orientation of genes in the gum gene cluster in *Xanthomonas campestris* pv. campestris!

Statistics

Theoretical background

- Normal distribution plot

Compare observed sample against the expected distribution

H0 = sample was taken from distribution

H0 can only be rejected or kept due to insufficient evidence against it

H0 can NEVER be confirmed!

Shapiro-Wilk test

scipy.stats.shapiro

scipy.stats.**shapiro**(*x*, *a=None*, *reta=False*) [source]

Perform the Shapiro-Wilk test for normality.

The Shapiro-Wilk test tests the null hypothesis that the data was drawn from a normal distribution.

Parameters:

x : *array_like*
Array of sample data.

a : *array_like, optional*
Array of internal parameters used in the calculation. If these are not given, they will be computed internally. If x has length n, then a must have length n/2.

reta : *bool, optional*
Whether or not to return the internally computed a values. The default is False.

Returns:

W : *float*
The test statistic.

p-value : *float*
The p-value for the hypothesis test.

a : *array_like, optional*
If *reta* is True, then these are the internally computed "a" values that may be passed into this function on future calls.

See also:

anderson — The Anderson-Darling test for normality

kstest — The Kolmogorov-Smirnov test for goodness of fit.

Notes

The algorithm used is described in [R634] but censoring parameters as described are not implemented. For N > 5000 the W test statistic is accurate but the p-value may not be.

The chance of rejecting the null hypothesis when it is true is close to 5% regardless of sample size.

https://docs.scipy.org

Correlation

scipy.stats.pearsonr

scipy.stats.pearsonr(*x*, *y*) [source]

Calculates a Pearson correlation coefficient and the p-value for testing non-correlation.

The Pearson correlation coefficient measures the linear relationship between two datasets. Strictly speaking, Pearson's correlation requires that each dataset be normally distributed. Like other correlation coefficients, this one varies between -1 and +1 with 0 implying no correlation. Correlations of -1 or +1 imply an exact linear relationship. Positive correlations imply that as x increases, so does y. Negative correlations imply that as x increases, y decreases.

The p-value roughly indicates the probability of an uncorrelated system producing datasets that have a Pearson correlation at least as extreme as the one computed from these datasets. The p-values are not entirely reliable but are probably reasonable for datasets larger than 500 or so.

Parameters: **x** : *(N,) array_like*
Input
y : *(N,) array_like*
Input

Returns: *(Pearson's correlation coefficient,*
2-tailed p-value)

https://docs.scipy.org

t-test

scipy.stats.ttest_ind

scipy.stats.**ttest_ind**(*a*, *b*, *axis=0*, *equal_var=True*, *nan_policy='propagate'*) [source]

Calculates the T-test for the means of *two independent* samples of scores.

This is a two-sided test for the null hypothesis that 2 independent samples have identical average (expected) values. This test assumes that the populations have identical variances by default.

Parameters:

a, b : *array_like*
The arrays must have the same shape, except in the dimension corresponding to *axis* (the first, by default).

axis : *int or None, optional*
Axis along which to compute test. If None, compute over the whole arrays, *a*, and *b*.

equal_var : *bool, optional*
If True (default), perform a standard independent 2 sample test that assumes equal population variances [R643]. If False, perform Welch's t-test, which does not assume equal population variance [R644].
New in version 0.11.0.

nan_policy : *{'propagate', 'raise', 'omit'}, optional*
Defines how to handle when input contains nan. 'propagate' returns nan, 'raise' throws an error, 'omit' performs the calculations ignoring nan values. Default is 'propagate'.

Returns:

statistic : *float or array*
The calculated t-statistic.

pvalue : *float or array*
The two-tailed p-value.

Notes

We can use this test, if we observe two independent samples from the same or different population, e.g. exam scores of boys and girls or of two ethnic groups. The test measures whether the average (expected) value differs significantly across samples. If we observe a large p-value, for example larger than 0.05 or 0.1, then we cannot reject the null hypothesis of identical average scores. If the p-value is smaller than the threshold, e.g. 1%, 5% or 10%, then we reject the null hypothesis of equal averages.

https://docs.scipy.org

Wilcoxon test

scipy.stats.wilcoxon

scipy.stats.wilcoxon(*x*, *y=None*, *zero_method='wilcox'*, *correction=False*) [source]

Calculate the Wilcoxon signed-rank test.

The Wilcoxon signed-rank test tests the null hypothesis that two related paired samples come from the same distribution. In particular, it tests whether the distribution of the differences x - y is symmetric about zero. It is a non-parametric version of the paired T-test.

Parameters:

x : *array_like*
The first set of measurements.

y : *array_like, optional*
The second set of measurements. If *y* is not given, then the *x* array is considered to be the differences between the two sets of measurements.

zero_method : *string, {"pratt", "wilcox", "zsplit"}, optional*

"pratt":
Pratt treatment: includes zero-differences in the ranking process (more conservative)

"wilcox":
Wilcox treatment: discards all zero-differences

"zsplit":
Zero rank split: just like Pratt, but spliting the zero rank between positive and negative ones

correction : *bool, optional*
If True, apply continuity correction by adjusting the Wilcoxon rank statistic by 0.5 towards the mean value when computing the z-statistic. Default is False.

Returns:

T : *float*
The sum of the ranks of the differences above or below zero, whichever is smaller.

p-value : *float*
The two-sided p-value for the test.

Notes

Because the normal approximation is used for the calculations, the samples used should be large. A typical rule is to require that n > 20.

Mann-Whitney rank test

scipy.stats.mannwhitneyu

scipy.stats. **mannwhitneyu** (*x*, *y*, *use_continuity=True*, *alternative=None*) [source]

Computes the Mann-Whitney rank test on samples x and y.

Parameters:

x, y : *array_like*
Array of samples, should be one-dimensional.

use_continuity : *bool, optional*
Whether a continuity correction (1/2.) should be taken into account. Default is True.

alternative : *None (deprecated), 'less', 'two-sided', or 'greater'*
Whether to get the p-value for the one-sided hypothesis ('less' or 'greater') or for the two-sided hypothesis ('two-sided'). Defaults to None, which results in a p-value half the size of the 'two-sided' p-value and a different U statistic. The default behavior is not the same as using 'less' or 'greater': it only exists for backward compatibility and is deprecated.

Returns:

statistic : *float*
The Mann-Whitney U statistic, equal to min(U for x, U for y) if *alternative* is equal to None (deprecated; exists for backward compatibility), and U for y otherwise.

pvalue : *float*
p-value assuming an asymptotic normal distribution. One-sided or two-sided, depending on the choice of *alternative*.

Notes

Use only when the number of observation in each sample is > 20 and you have 2 independent samples of ranks. Mann-Whitney U is significant if the u-obtained is LESS THAN or equal to the critical value of U.

This test corrects for ties and by default uses a continuity correction.

https://docs.scipy.org

Chi square test

scipy.stats.chisquare

scipy.stats.**chisquare**(*f_obs*, *f_exp=None*, *ddof=0*, *axis=0*) [source]

Calculates a one-way chi square test.

The chi square test tests the null hypothesis that the categorical data has the given frequencies.

Parameters:

f_obs : *array_like*
Observed frequencies in each category.

f_exp : *array_like, optional*
Expected frequencies in each category. By default the categories are assumed to be equally likely.

ddof : *int, optional*
"Delta degrees of freedom": adjustment to the degrees of freedom for the p-value. The p-value is computed using a chi-squared distribution with `k - 1 - ddof` degrees of freedom, where *k* is the number of observed frequencies. The default value of *ddof* is 0.

axis : *int or None, optional*
The axis of the broadcast result of *f_obs* and *f_exp* along which to apply the test. If axis is None, all values in *f_obs* are treated as a single data set. Default is 0.

Returns:

chisq : *float or ndarray*
The chi-squared test statistic. The value is a float if *axis* is None or *f_obs* and *f_exp* are 1-D.

p : *float or ndarray*
The p-value of the test. The value is a float if *ddof* and the return value *chisq* are scalars.

See also:
`power_divergence`, `mstats.chisquare`

Notes

This test is invalid when the observed or expected frequencies in each category are too small. A typical rule is that all of the observed and expected frequencies should be at least 5.

The default degrees of freedom, k-1, are for the case when no parameters of the distribution are estimated. If p parameters are estimated by efficient maximum likelihood then the correct degrees of freedom are k-1-p. If the parameters are estimated in a different way, then the dof can be between k-1-p and k-1. However, it is also possible that the asymptotic distribution is not a chisquare, in which case this test is not appropriate.

https://docs.scipy.org

Exercise: analyze the unknown data

- Construct a suitable visualization!
- Analyze distribution and trends!
- Apply statistical test to investigate difference!

HTML

- Construction of a HTML-based heatmap

gene ID	0H	4H	Salt	Heat	Inflorescence	leaf	Root	seedlings_HiK	seedlings_HiK2	Leaf_SSC
(p	0.02	0.0	0.0	0.0	0.0	0.0	4.33	1.44	1.2	0.0
(c	0.0	0.0	0.0	0.0	0.0	0.0	1.17	0.11	0.0	0.0
(c	0.0	0.0	0.0	0.0	0.0	0.0	3.0	0.11	0.4	0.0
p	1.76	29.23	14.65	6.11	203.75	265.2	0.83	199.33	199.7	54.4
o	0.0	0.0	0.0	0.01	1.38	0.0	0.0	0.0	0.1	0.0
(	0.0	0.03	0.01	0.0	0.25	0.0	0.0	0.0	0.1	0.02
y	0.01	0.01	0.0	0.06	13.88	0.0	0.0	0.11	0.1	0.01
u	6.25	58.89	41.42	15.61	69.5	5.4	3.83	157.22	163.1	22.49
h	0.86	5.04	0.09	0.24	33.5	0.8	12.67	8.78	8.5	1.07
y	10.74	43.0	4.78	6.45	29.0	1.2	7.17	62.67	67.0	25.48
e	19.29	14.22	6.14	5.21	4.88	1.6	11.5	28.11	24.2	13.68
w	35.33	45.16	52.79	70.1	17.5	74.2	6.67	21.89	22.6	65.95
q	6.03	24.4	2.69	2.38	80.0	35.6	19.83	55.22	59.9	11.16
k	5.74	15.9	1.35	4.56	57.5	3.0	1.17	9.44	7.5	8.47
s	0.27	1.74	0.07	0.25	11.38	0.2	0.0	0.78	0.4	0.11
i	0.04	0.0	0.0	0.0	36.0	0.0	0.0	0.89	1.1	0.0
	0.0	0.0	0.0	0.0	18.0	0.0	0.0	0.67	0.7	0.0
	9.84	14.92	17.27	7.68	16.13	27.0	26.0	26.89	23.7	11.58
ev	17.99	9.41	7.94	8.45	11.38	21.2	13.83	19.22	19.7	14.81

HTML template

different conditions/tissues

different genes/transcripts

```
<table border=1>
    <tr>
        {% for name in column_names %}
            <th> {{ name }} </th>
        {% endfor %}
    </tr>
    {% for gene in genes %}
        <tr width="20px">
            <th>{{ gene['name'] }}</th>
            {% for value in gene['values'] %}
                <td bgcolor={{value['color']}} width="20px"> <span title={{value['title']}}> <font color="black">  {{value['value']}} </font> </span></td>
                <!-- one field contains the absolute value and has an corresponding background color -->
            {% endfor %}
        </tr>
    {% endfor %}
</table>
```

precomputed color

value converted to test

construct_heatmap.py reads values from text file and prepares data structures to fill this template

HTML document can be converted to PDF

Exercise: construct heatmap

- Read data table and construct heatmap for the gene expression in HTML!
- Add mouse-over effect to display functional gene annotation!

Biological background of presented examples

- Nd-1 genome assembly (Pucker *et al.*, 2016):
  - https://doi.org/10.1371/journal.pone.0164321
- Non-canonical splice sites (Pucker *et al.*, 2017):
  - https://doi.org/10.1186/s13104-017-2985-y
- *Croton tiglium* transcriptome assembly (Haak *et al.*, 2018):
  - https://doi.org/10.3389/fmolb.2018.00062