Introduction to

with Application to Bioinformatics

- Day 2

Review Day 1

Go to Canvas, `Modules -> Day 2 -> Review Day 1`

~30 minutes

Variables and Types

1. Which of the following is of the type `float`?:

7.4

2. Match the following variables with their type:

`var1 = 54`	**integer**
`var2 = [1,2,7,1,24]`	**list**
`var3 = 2.98`	**float**
`var4 = True`	**boolean**

Literals

All literals have a type:

- Strings (str) ‘Hello’ “Hi”
- Integers (int) 5
- Floats (float) 3.14
- Boolean (bool) True or False

In [7]:
```
type(True)
```

Out[7]:
```
bool
```

Variables

Used to store values and to assign them a name.

In [8]:
```
a = 3.14
a
```

Out[8]:
```
3.14
```

Variables

Used to store values and to assign them a name.

In [8]:
```
a = 3.14
a
```

Out[8]:
```
3.14
```

Lists

A collection of values.

In [9]:
```
x = [1,5,3,7,8]
y = ['a','b','c']
type(y)
```

Out[9]:
```
list
```

Comments

3. Which of the following symbols can be used to write comments in your code?

#

Operations

4. What happens if you do `[1,2,5,11] + [87,2,43,3]`?

`[1,2,5,11,87,2,43,3]` The lists will be concatenated

5. How do you find out if the variable `x` is present in a the list `mylist`?

Two answers correct:

1. `x in mylist`
2.
```
for l in mylist:
   if l == x:
       print('Found a
match')
```

6. How do you find out if 5 is larger than 3 and the integer 4 is the same as the float 4? Fill in all the missing code.

`5 > 3 and 4 == 4.0`

Basic operations

Type	Operations
int	+ - / ** % // ...
float	+ - / * % // ...
string	+

In [10]:

```
a = 2
b = 5.46
c = [1,2,3,4]
d = [5,6,7,8]
e = 7

e * a
```

Out[10]:

```
14
```

Comparison/Logical/Membership operators

Operation	Meaning
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
==	equal
!=	not equal

Operation	Meaning
and	connects two statements, both conditions having to be fulfilled
or	connects two statements, either conditions having to be fulfilled
not	reverses and/or

Operation	Meaning
in	value in object
not in	value not in object

In [11]:

```
a = [1,2,3,4,5,6,7,8]
b = 5
c = 10
b in a
b < c and c == 1
b not in a
```

Out[11]:

```
False
```

Sequences

7. How do you select the second element in the variable `mylist = [4,3,8,10]`?

`mylist[1]`

8. Pair the following variables with whether they are mutable or immutable

`var1 = 'my pretty string'`	**immutable**
`var2 = [1,2,3,4,5]`	**mutable**
`var3 = "hello world"`	**immutable**
`var4 = ['a', 'b', 'c', 'd']`	**mutable**

9. Which of the following types are iterable?

Lists and strings

Indexing

Lists (and strings) are an ORDERED collection of elements where every element can be access through an index.

`a[0]` : first item in list a

REMEMBER! Indexing starts at 0 in python

In [12]:

```
a = [1,2,3,4,5]
b = ['a','b','c']
c = 'a random string'

c[2]
c[1:4]
```

Out[12]:

```
' ra'
```

Mutable / Immutable sequences and iterables

Lists are mutable object, meaning you can use an index to change the list, while strings are immutable and therefore not changeable.

An iterable sequence is anything you can loop over, ie, lists and strings.

In [13]:

```
a = [1,2,3,4,5]         # mutable
b = ['a','b','c']       # mutable
c = 'a random string'   # immutable

c[0] = 'A'
a[0] = 42
a
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [13], in <cell line: 5>()
      2 b = ['a','b','c']       # mutable
      3 c = 'a random string'   # immutable
----> 5 c[0] = 'A'
      6 a[0] = 42
      7 a

TypeError: 'str' object does not support item assignment
```

New data type: `tuples`

- A tuple is an immutable sequence of objects
- Unlike a list, nothing can be changed in a tuple
- Still iterable

In [14]:

```
myTuple = (1,2,3,4,'a','b','c',[42,43,44])
#myTuple[0] = 42
#print(myTuple)
#print(len(myTuple))
for i in myTuple:
     print(i)
```

```
1
2
3
4
a
b
c
[42, 43, 44]
```

If/ Else statements

10. How do you do to print ‘Yes’ if x is bigger than y?

```
if x > y:
  print('Yes')
```

If/ Else statements

10. How do you do to print 'Yes' if x is bigger than y?

```
if x > y:
  print('Yes')
```

In [15]:

```
a = 3
b = [1,2,3,4]
if a in b:
    print(str(a)+' is found in the list b')
else:
    print(str(a)+' is not in the list')
```

```
3 is found in the list b
```

Files and loops

How do you open a file handle to read a file called ‘somerandomfile.txt’?

```
fh = open('somerandomfile.txt')
```

The file in the previous question contains several lines, how do you print each line?

1.
```
for line in fh:
    print(line)
```

2.
```
for row in fh:
    print(row)
```

In [16]:

```
fh = open('../files/somerandomfile.txt','r', encoding = 'utf-8')
for line in fh:
    print(line.strip())
fh.close()
```

```
just a strange
file with
some
nonsense lines
```

In [17]:

```
numbers = [5,6,7,8]
i = 0
while i < len(numbers):
    print(numbers[i])
    i += 1
```

```
5
6
7
8
```

Questions?

Day 2

- Pseudocode
- Functions vs Methods

How to approach a coding task

Problem:
You have a VCF file with a larger number of samples. You are interested in only one of the samples (sample1) and one region (chr5, 1.000.000-1.005.000). What you want to know is whether this sample has any variants in this region, and if so, what variants.

Always write pseudocode!

Pseudocode is a description of what you want to do without actually using proper syntax

What is your input?

A VCF file that is iterable

```
##ALT=<ID=NON_REF,Description="Represents any possible alternative allele at this location">
##FILTER=<ID=LOW_VQSLOD,Description="VQSLOD < 0.0">
##FILTER=<ID=LowQual,Description="Low quality">
##source=SelectVariants
##bcftools_mergeVersion=1.5+htslib-1.5
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  sample1 sample2 sample3
1       10492   .       C       T       550.31  PASS      AN=26;AC=2      GT:AD:DP  0/0:0,25:25       0/1:14,23:37       1/1:31,0:31
```

What is your input?

A VCF file that is iterable

```
##ALT=<ID=NON_REF,Description="Represents any possible alternative allele at this location">
##FILTER=<ID=LOW_VQSLOD,Description="VQSLOD < 0.0">
##FILTER=<ID=LowQual,Description="Low quality">
##source=SelectVariants
##bcftools_mergeVersion=1.5+htslib-1.5
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  sample1 sample2 sample3
1       10492   .       C       T       550.31  PASS      AN=26;AC=2      GT:AD:DP  0/0:0,25:25       0/1:14,23:37       1/1:31,0:31
```

Basic Pseudocode:

- Open file and loop over lines (ignore lines with #)
- Identify lines where chromosome is 5 and position is between 1.000.000 and 1.005.000
- Isolate the column that contains the genotype for sample1
- Extract the genotypes only from the column
- Check if the genotype contains any alternate alleles
- Print any variants containing alternate alleles for this sample between specified region

```
##ALT=<ID=NON_REF,Description="Represents any possible alternative allele at this location">
##FILTER=<ID=LOW_VQSLOD,Description="VQSLOD < 0.0">
##FILTER=<ID=LowQual,Description="Low quality">
##source=SelectVariants
##bcftools_mergeVersion=1.5+htslib-1.5
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  sample1 sample2 sample3
1       10492   .       C       T       550.31  PASS      AN=26;AC=2      GT:AD:DP  0/0:0,25:25       0/1:14,23:37       1/1:31,0:31
```

- Open file and loop over lines (ignore lines starting with #)

In [18]:

```
fh = open('/mnt/c/Users/Nina/Documents/courses/Python_Beginner_Course/genotypes.vcf', 'r', encoding = 'utf-8')
for line in fh:
    if not line.startswith('#'):
        print(line.strip())
        break
fh.close()
# Next, find chromosome 5
```

```
1       10492   .       C       T       550.31  LOW_VQSLOD      AN=26;AC=2      GT:AD:DP:GQ:PGT:PID:PL  ./.:0,0:
0:.:.:.:.       ./.:0,0:0:.:.:.:.       ./.:0,0:0:.:.:.:.       ./.:0,0:0:.:.:.:.       ./.:0,0:0:.:.:.:.       0/1:12,
7:19:99:0|1:10403_ACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAAC_A:196,0,340     ./.:0,0:0:.:.:.:.       ./.:0,0:0:.:.:.:.
./.:0,0:0:.:.:.:.       ./.:0,0:0:.:.:.:.       0/1:18,4:22:48:.:.:48,0,504     ./.:0,0:0:.:.:.:.       ./.:0,0:
0:.:.:.:.
```

- Identify lines where chromosome is 5 and position is between 1.000.000 and 1.005.000

```
##ALT=<ID=NON_REF,Description="Represents any possible alternative allele at this location">
##FILTER=<ID=LOW_VQSLOD,Description="VQSLOD < 0.0">
##FILTER=<ID=LowQual,Description="Low quality">
##source=SelectVariants
##bcftools_mergeVersion=1.5+htslib-1.5
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  sample1 sample2 sample3
1       10492   .       C       T       550.31  PASS      AN=26;AC=2      GT:AD:DP  0/0:0,25:25       0/1:14,23:37       1/1:31,0:31
```

In [19]:

```
fh = open('/mnt/c/Users/Nina/Documents/courses/Python_Beginner_Course/genotypes.vcf', 'r', encoding = 'utf-8')
for line in fh:
    if not line.startswith('#'):
        cols = line.strip().split('\t')
        if cols[0] == '5':
            print(cols)
            break
fh.close()

# Next, find the correct region
```

```
['5', '12041', '.', 'A', 'T', '18075.2', 'PASS', 'AN=26;AC=2', 'GT:AD:DP:GQ:PL', './.:0,0:0:.:.', './.:0,0:0:.:.', './.:0,0:0:.:.', './.:0,0:0:.:.', './.:0,0:0:.:.', './.:0,0:0:.:.', './.:0,0:0:.:.', './.:0,0:0:.:.', './.:0,0:0:.:.', '0/1:15,6:21:99:142,0,391', './.:0,0:0:.:.', '0/1:16,17:33:99:442,0,422']
```

```
##ALT=<ID=NON_REF,Description="Represents any possible alternative allele at this location">
##FILTER=<ID=LOW_VQSLOD,Description="VQSLOD < 0.0">
##FILTER=<ID=LowQual,Description="Low quality">
##source=SelectVariants
##bcftools_mergeVersion=1.5+htslib-1.5
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  sample1 sample2 sample3
1       10492   .       C       T       550.31  PASS      AN=26;AC=2      GT:AD:DP  0/0:0,25:25       0/1:14,23:37       1/1:31,0:31
```

In [20]:

```
fh = open('/mnt/c/Users/Nina/Documents/courses/Python_Beginner_Course/genotypes.vcf', 'r', encoding = 'utf-8')
for line in fh:
    if not line.startswith('#'):
        cols = line.strip().split('\t')
        if cols[0] == '5' and \
           int(cols[1]) >= 1000000 and int(cols[1]) <= 1005000:
                print(cols)
                break
fh.close()
# Next, find the genotypes for sample1
```

```
['5', '1000080', '.', 'A', 'T', '2557.1', 'PASS', 'AN=26;AC=2', 'GT:AD:DP:GQ:PL', '0/1:15,18:33:99:489,0,357', './.:0,0:
0:.:.', './.:0,0:0:.:.', './.:0,0:0:.:.', './.:0,0:0:.:.', './.:0,0:0:.:.', './.:0,0:0:.:.', './.:0,0:0:.:.', '0/1:21,1
9:40:99:481,0,542', './.:0,0:0:.:.', './.:0,0:0:.:.', './.:0,0:0:.:.', './.:0,0:0:.:.']
```

- Isolate the column that contains the genotype for sample1

```
##ALT=<ID=NON_REF,Description="Represents any possible alternative allele at this location">
##FILTER=<ID=LOW_VQSLOD,Description="VQSLOD < 0.0">
##FILTER=<ID=LowQual,Description="Low quality">
##source=SelectVariants
##bcftools_mergeVersion=1.5+htslib-1.5
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  sample1 sample2 sample3
1       10492   .       C       T       550.31  PASS      AN=26;AC=2      GT:AD:DP  0/0:0,25:25       0/1:14,23:37       1/1:31,0:31
```

In [21]:

```
fh = open('/mnt/c/Users/Nina/Documents/courses/Python_Beginner_Course/genotypes.vcf', 'r', encoding = 'utf-8')
for line in fh:
    if not line.startswith('#'):
        cols = line.strip().split('\t')
        if cols[0] == '5' and \
           int(cols[1]) >= 1000000 and int(cols[1]) <= 1005000:
                geno = cols[9]
                print(geno)
                break
fh.close()
# Next, extract the genotypes only
```

```
0/1:15,18:33:99:489,0,357
```

- Extract the genotypes only from the column

```
##ALT=<ID=NON_REF,Description="Represents any possible alternative allele at this location">
##FILTER=<ID=LOW_VQSLOD,Description="VQSLOD < 0.0">
##FILTER=<ID=LowQual,Description="Low quality">
##source=SelectVariants
##bcftools_mergeVersion=1.5+htslib-1.5
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  sample1 sample2 sample3
1       10492   .       C       T       550.31  PASS    AN=26;AC=2      GT:AD:DP  0/0:0,25:25       0/1:14,23:37       1/1:31,0:31
```

In [22]:

```
fh = open('/mnt/c/Users/Nina/Documents/courses/Python_Beginner_Course/genotypes.vcf', 'r', encoding = 'utf-8')
for line in fh:
    if not line.startswith('#'):
        cols = line.strip().split('\t')
        if cols[0] == '5' and \
           int(cols[1]) >= 1000000 and int(cols[1]) <= 1005000:
                geno = cols[9].split(':')[0]
                print(geno)
                break
fh.close()
# Next, find in which positions sample1 has alternate alleles
```

```
0/1
```

- Check if the genotype contains any alternate alleles

```
##ALT=<ID=NON_REF,Description="Represents any possible alternative allele at this location">
##FILTER=<ID=LOW_VQSLOD,Description="VQSLOD < 0.0">
##FILTER=<ID=LowQual,Description="Low quality">
##source=SelectVariants
##bcftools_mergeVersion=1.5+htslib-1.5
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  sample1 sample2 sample3
1       10492   .       C       T       550.31  PASS    AN=26;AC=2      GT:AD:DP  0/0:0,25:25       0/1:14,23:37       1/1:31,0:31
```

In [23]:

```
fh = open('/mnt/c/Users/Nina/Documents/courses/Python_Beginner_Course/genotypes.vcf', 'r', encoding = 'utf-8')
for line in fh:
    if not line.startswith('#'):
        cols = line.strip().split('\t')
        if cols[0] == '5' and \
           int(cols[1]) >= 1000000 and int(cols[1]) <= 1005000:
                geno = cols[9].split(':')[0]
                if geno in ['0/1', '1/1']:
                    print(geno)
fh.close()
#Next, print nicely
```

```
0/1
0/1
0/1
0/1
0/1
0/1
0/1
0/1
0/1
0/1
0/1
0/1
0/1
0/1
0/1
0/1
0/1
0/1
0/1
0/1
```

```
0/1
0/1
0/1
```

- Print any variants containing alternate alleles for this sample between specified region

```
##ALT=<ID=NON_REF,Description="Represents any possible alternative allele at this location">
##FILTER=<ID=LOW_VQSLOD,Description="VQSLOD < 0.0">
##FILTER=<ID=LowQual,Description="Low quality">
##source=SelectVariants
##bcftools_mergeVersion=1.5+htslib-1.5
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  sample1 sample2 sample3
1       10492   .       C       T       550.31  PASS      AN=26;AC=2      GT:AD:DP  0/0:0,25:25       0/1:14,23:37       1/1:31,0:31
```

In [24]:

```
fh = open('/mnt/c/Users/Nina/Documents/courses/Python_Beginner_Course/genotypes.vcf', 'r', encoding = 'utf-8')
res = []
for line in fh:
    if not line.startswith('#'):
        cols = line.strip().split('\t')
        if cols[0] == '5' and \
           int(cols[1]) >= 1000000 and int(cols[1]) <= 1005000:
                geno = cols[9].split(':')[0]
                if geno in ['0/1', '1/1']:
                    var = cols[0]+':'+cols[1]+'_'+cols[3]+'-'+cols[4]
                 #   print(var+' has genotype: '+geno)
                    res.append(var)
fh.close()
print(res)
```

```
['5:1000080_A-T', '5:1000156_G-A', '5:1001097_C-A', '5:1001193_C-T', '5:1001245_T-C', '5:1001339_C-T', '5:1001344_G-C',
'5:1001683_G-T', '5:1001755_G-A', '5:1002374_G-A', '5:1002382_G-C', '5:1002620_T-C', '5:1002722_G-A', '5:1002819_C-A',
'5:1003043_G-T', '5:1003099_C-T', '5:1003135_G-A', '5:1004648_A-G', '5:1004650_A-C', '5:1004665_A-G', '5:1004702_G-T',
'5:1004879_T-C']
```

→ **Exercises Day 2**

3 options:

1. <p style="color:green";>Green exercise</p>
2. <p style="color:#FFBF00";>Yellow exercise</p>
3. <p style="color:red";>Red exercise</p>

Level of complexity increases with each exercises
New to programming: Do Green exercise and possibly Yellow exercise
More experienced: Do Yellow exercise and/or Red exercise

More useful functions and methods

What is the difference between a `function` and a `method`?

A `method` always belongs to an object of a specific class, a `function` does not have to. For example:

`print('a string')` and `print(42)` both works, even though one is a string and one is an integer

`'a string '.strip()` works, but `[1,2,3,4].strip()` does not work. `strip()` is a method that only works on strings

What does it matter to me?

For now, you mostly need to be aware of the difference, and know the different syntaxes:

A function:
`functionName()`

A method:
`<object>.methodName()`

In [25]:

```
len([1,2,3])
len('a string')

'a string  '.strip()
[1,2,3].strip()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [25], in <cell line: 5>()
      2 len('a string')
      4 'a string  '.strip()
----> 5 [1,2,3].strip()

AttributeError: 'list' object has no attribute 'strip'
```

Functions

Built-in Functions				
abs( )	delattr( )	hash( )	memoryview( )	set( )
all( )	dict( )	help( )	min( )	setattr( )
any( )	dir( )	hex( )	next( )	slice( )
ascii( )	divmod( )	id( )	object( )	sorted( )
bin( )	enumerate( )	input( )	oct( )	staticmethod( )
bool( )	eval( )	int( )	open( )	str( )
breakpoint( )	exec( )	isinstance( )	ord( )	sum( )
bytearray( )	filter( )	issubclass( )	pow( )	super( )
bytes( )	float( )	iter( )	print( )	tuple( )
callable( )	format( )	len( )	property( )	type( )
chr( )	frozenset( )	list( )	range( )	vars( )
classmethod( )	getattr( )	locals( )	repr( )	zip( )
compile( )	globals( )	map( )	reversed( )	__import__( )
complex( )	hasattr( )	max( )	round( )	

Python Built-in functions

Built-in Functions				
abs( )	delattr( )	hash( )	memoryview( )	set( )
all( )	dict( )	help( )	min( )	setattr( )
any( )	dir( )	hex( )	next( )	slice( )
ascii( )	divmod( )	id( )	object( )	sorted( )
bin( )	enumerate( )	input( )	oct( )	staticmethod( )
bool( )	eval( )	int( )	open( )	str( )
breakpoint( )	exec( )	isinstance( )	ord( )	sum( )
bytearray( )	filter( )	issubclass( )	pow( )	super( )
bytes( )	float( )	iter( )	print( )	tuple( )
callable( )	format( )	len( )	property( )	type( )
chr( )	frozenset( )	list( )	range( )	vars( )
classmethod( )	getattr( )	locals( )	repr( )	zip( )
compile( )	globals( )	map( )	reversed( )	__import__( )
complex( )	hasattr( )	max( )	round( )	

In [26]:

```
range(5)
```

Out[26]:

```
range(0, 5)
```

Built-in Functions				
abs( )	delattr( )	hash( )	memoryview( )	set( )
all( )	dict( )	help( )	**min( )**	setattr( )
any( )	dir( )	hex( )	next( )	slice( )
ascii( )	divmod( )	id( )	object( )	**sorted( )**
bin( )	enumerate( )	input( )	oct( )	staticmethod( )
bool( )	eval( )	int( )	open( )	str( )
breakpoint( )	exec( )	isinstance( )	ord( )	**sum( )**
bytearray( )	filter( )	issubclass( )	pow( )	super( )
bytes( )	float( )	iter( )	print( )	tuple( )
callable( )	format( )	**len( )**	property( )	type( )
chr( )	frozenset( )	list( )	range( )	vars( )
classmethod( )	getattr( )	locals( )	repr( )	zip( )
compile( )	globals( )	map( )	reversed( )	__import__( )
complex( )	hasattr( )	**max( )**	round( )	

In [27]:

```
sorted([1,2,35,23,88,4])
```

Out[27]:

```
[1, 2, 4, 23, 35, 88]
```

From Python documentation

sum(*iterable*[, *start*])

Sums *start* and the items of an *iterable* from left to right and returns the total. *start* defaults to `0`. The *iterable*'s items are normally numbers, and the start value is not allowed to be a string.

In [28]:

```
sum([1,2,3,4],10)
help(sum)
```

```
Help on built-in function sum in module builtins:

sum(iterable, /, start=0)
    Return the sum of a 'start' value (default: 0) plus an iterable of numbers

    When the iterable is empty, return the start value.
    This function is intended specifically for use with numeric values and may
    reject non-numeric types.
```

Built-in Functions				
abs( )	delattr( )	hash( )	memoryview( )	set( )
all( )	dict( )	help( )	min( )	setattr( )
any( )	dir( )	hex( )	next( )	slice( )
ascii( )	divmod( )	id( )	object( )	sorted( )
bin( )	enumerate( )	input( )	oct( )	staticmethod( )
bool( )	eval( )	int( )	open( )	str( )
breakpoint( )	exec( )	isinstance( )	ord( )	sum( )
bytearray( )	filter( )	issubclass( )	pow( )	super( )
bytes( )	float( )	iter( )	print( )	tuple( )
callable( )	format( )	len( )	property( )	type( )
chr( )	frozenset( )	list( )	range( )	vars( )
classmethod( )	getattr( )	locals( )	repr( )	zip( )
compile( )	globals( )	map( )	reversed( )	__import__( )
complex( )	hasattr( )	max( )	round( )	

In [29]:

```
round(3.234556, 3)
a = 'my string'
list(a)
```

Out[29]:

```
['m', 'y', ' ', 's', 't', 'r', 'i', 'n', 'g']
```

Methods

Useful operations on strings

String Methods	
str.strip( )	str.startswith( )
str.rstrip( )	str.endswith( )
str.lstrip( )	str.upper( )
str.split( )	str.lower( )
str.join( )	

Methods

Useful operations on strings

String Methods	
str.strip( )	str.startswith( )
str.rstrip( )	str.endswith( )
str.lstrip( )	str.upper( )
str.split( )	str.lower( )
str.join( )	

`str.`**`strip`**([*chars*])

Return a copy of the string with the leading and trailing characters removed. The *chars* argument is a string specifying the set of characters to be removed. If omitted or `None`, the *chars* argument defaults to removing whitespace. The *chars* argument is not a prefix or suffix; rather, all combinations of its values are stripped:

```
>>> '   spacious   '.strip()
'spacious'
>>> 'www.example.com'.strip('cmowz.')
'example'
```

str.**lstrip**([*chars*])

Return a copy of the string with leading characters removed. The *chars* argument is a string specifying the set of characters to be removed. If omitted or `None`, the *chars* argument defaults to removing whitespace. The *chars* argument is not a prefix; rather, all combinations of its values are stripped:

```
>>> '   spacious   '.lstrip()
'spacious   '
>>> 'www.example.com'.lstrip('cmowz.')
'example.com'
```

str.**rstrip**([*chars*])

Return a copy of the string with trailing characters removed. The *chars* argument is a string specifying the set of characters to be removed. If omitted or `None`, the *chars* argument defaults to removing whitespace. The *chars* argument is not a suffix; rather, all combinations of its values are stripped:

```
>>> '   spacious   '.rstrip()
'   spacious'
>>> 'mississippi'.rstrip('ipz')
'mississ'
```

In [30]:

```
'    spaciou   sWith5678.com\n'.strip()
```

Out[30]:

```
'spaciou   sWith5678.com'
```

`str.`**`split`**(*sep=None*, *maxsplit=-1*)

Return a list of the words in the string, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done (thus, the list will have at most `maxsplit+1` elements). If *maxsplit* is not specified or `-1`, then there is no limit on the number of splits (all possible splits are made).

If *sep* is given, consecutive delimiters are not grouped together and are deemed to delimit empty strings (for example, `'1,,2'.split(',')` returns `['1', '', '2']`). The *sep* argument may consist of multiple characters (for example, `'1<>2<>3'.split('<>')` returns `['1', '2', '3']`). Splitting an empty string with a specified separator returns `['']`.

For example:

```
>>> '1,2,3'.split(',')
['1', '2', '3']
>>> '1,2,3'.split(',', maxsplit=1)
['1', '2,3']
>>> '1,2,,3,'.split(',')
['1', '2', '', '3', '']
```

If *sep* is not specified or is `None`, a different splitting algorithm is applied: runs of consecutive whitespace are regarded as a single separator, and the result will contain no empty strings at the start or end if the string has leading or trailing whitespace. Consequently, splitting an empty string or a string consisting of just whitespace with a `None` separator returns `[]`.

For example:

```
>>> '1 2 3'.split()
['1', '2', '3']
>>> '1 2 3'.split(maxsplit=1)
['1', '2 3']
>>> '   1   2   3   '.split()
['1', '2', '3']
```

In [31]:

```
a = '  split a string into a list '
a.split(maxsplit=3)
```

Out[31]:

```
['split', 'a', 'string', 'into a list ']
```

`str.`**`join`**(*iterable*)

Return a string which is the concatenation of the strings in *iterable*. A `TypeError` will be raised if there are any non-string values in *iterable*, including `bytes` objects. The separator between elements is the string providing this method.

In [32]:

```
'|'.join('a string already')
''.join(['a', 'b', 'c', 'd'])
' '.join([1,2,3])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [32], in <cell line: 3>()
      1 '|'.join('a string already')
      2 ''.join(['a', 'b', 'c', 'd'])
----> 3 ' '.join([1,2,3])

TypeError: sequence item 0: expected str instance, int found
```

`str.` **`startswith`**(*prefix*[, *start*[, *end*]])

Return `True` if string starts with the *prefix*, otherwise return `False`. *prefix* can also be a tuple of prefixes to look for. With optional *start*, test string beginning at that position. With optional *end*, stop comparing string at that position.

`str.` **`endswith`**(*suffix*[, *start*[, *end*]])

Return `True` if the string ends with the specified *suffix*, otherwise return `False`. *suffix* can also be a tuple of suffixes to look for. With optional *start*, test beginning at that position. With optional *end*, stop comparing at that position.

In [33]:

```
'long string'.startswith('ng',2)
#'long string'.endswith('nt')
```

Out[33]:

```
True
```

`str.`**`upper`**`()`

Return a copy of the string with all the cased characters [4] converted to uppercase. Note that `s.upper().isupper()` might be `False` if `s` contains uncased characters or if the Unicode category of the resulting character(s) is not “Lu” (Letter, uppercase), but e.g. “Lt” (Letter, titlecase).

`str.`**`lower`**`()`

Return a copy of the string with all the cased characters [4] converted to lowercase.

In [34]:

```
'LongRandomString'.lower()
'LongRandomString'.upper()
```

Out[34]:

```
'LONGRANDOMSTRING'
```

Useful operations on Mutable sequences

Operation	Result
s.append(x)	appends *x* to the end of the sequence
s.insert(i, x)	*x* is inserted at pos *i*
s.pop([i])	retrieves the item *i* from *s* and also removes it
s.remove(x)	retrieves the first item from *s* where s[i] == x
s.reverse( )	reverses the items of *s* in place

In [35]:

```
a = [1,2,3,4,5,5,5,5]
a.append(6)
a.pop(2)
a.reverse()
a.remove(5)

b = (1,2,3,4)
c = [1,2,3,4]
c.append(5)
c
```

Out[35]:

```
[1, 2, 3, 4, 5]
```

Summary

- Tuples are immutable sequences of objects
- Always plan your approach before you start coding
- A method always belongs to an object of a specific class, a function does not have to
- The official Python documentation describes the syntax for all built-in functions and methods

→ **Exercises Day 2**

3 options:

1. <p style="color:green";>Green exercise</p>
2. <p style="color:#FFBF00";>Yellow exercise</p>
3. <p style="color:red";>Red exercise</p>

Level of complexity increases with each exercises
New to programming: Do Green exercise and possibly Yellow exercise
More experienced: Do Yellow exercise and/or Red exercise

IMDb

Download the 250.imdb file from the course website

This format of this file is:

- Line by line
- Columns separated by the | character
- Header starting with #

```
# Votes | Rating | Year | Runtime | URL | Genres | Title
   126807|    8.5|1957|5280|https://images-na.ssl-images....|Drama,War|Paths of Glory
    71379|    8.2|1925|4320|https://images-na.ssl-images....|Adventure,Comedy,Drama,Family|The Gold
```

Votes | Rating | Year | Runtime | URL | Genres | Title

Find the movie with the highest rating

```
# Votes | Rating | Year | Runtime | URL | Genres | Title
   126807|    8.5|1957|5280|https://images-na.ssl-images....|Drama,War|Paths of Glory
    71379|    8.2|1925|4320|https://images-na.ssl-images....|Adventure,Comedy,Drama,Family|The Gold
```

Write step-by-step pseudocode

```
# Votes | Rating | Year | Runtime | URL | Genres | Title
   126807|   8.5|1957|5280|https://images-na.ssl-images....|Drama,War|Paths of Glory
    71379|   8.2|1925|4320|https://images-na.ssl-images....|Adventure,Comedy,Drama,Family|The Gold
```

- Open file
- Initiate counter to keep track of highest rating and movie. Start counter at 0
- Loop over all lines not starting with '#'
- Strip and split the lines into a list
- Save the element containing the rating from the list into a variable
- If current rating is higher than the rating in the counter, replace counter value with rating and movie
- Close file
- Print counter

```
# Votes | Rating | Year | Runtime | URL | Genres | Title
   126807|   8.5|1957|5280|https://images-na.ssl-images....|Drama,War|Paths of Glory
    71379|   8.2|1925|4320|https://images-na.ssl-images....|Adventure,Comedy,Drama,Family|The Gold
```

In [36]:

```
fh   = open('../downloads/250.imdb', 'r', encoding = 'utf-8')
best = [0,'']           # here we save the rating and which movie
for line in fh:
    if not line.startswith('#'):
        cols   = line.strip().split('|')
        rating = float(cols[1].strip())
        if rating > best[0]:           # if the rating is higher than previous highest, update best
            best = [rating,cols[6]]
fh.close()
print(best)
```

```
[9.3, 'The Shawshank Redemption']
```