Introduction to

python

with Application to Bioinformatics

- Day 4

Start by doing today's quiz (Review Day 3)

Review: In what ways does the type of an object matter?

In [30]:

```
row = 'sofa|2000|buy|Uppsala'
fields = row.split('|')
price = (fields[1])
if price == 2000:
    print('The price is a number!')
if price == '2000':
    print('The price is a string!')
```

```
The price is a string!
```

Review: In what ways does the type of an object matter?

In [30]:

```
row = 'sofa|2000|buy|Uppsala'
fields = row.split('|')
price = (fields[1])
if price == 2000:
    print('The price is a number!')
if price == '2000':
    print('The price is a string!')
```

```
The price is a string!
```

In [31]:

```
print(sorted([ 2000,   30,   100 ]))
print(sorted(['2000', '30', '100']))
# Hint: is `'30' > '2000'`?
```

```
[30, 100, 2000]
['100', '2000', '30']
```

In what ways does the type of an object matter?

- Each type store a specific type of information
  - `int` for integers,
  - `float` for floating point values (decimals),
  - `str` for strings,
  - `list` for lists,
  - `dict` for dictionaries.
- Each type supports different operations, functions and methods.

- Each type supports different **operations**, functions and methods

```
In [32]: 30 > 2000
```

```
Out[32]: False
```

```
In [33]: '30' > '2000'
```

```
Out[33]: True
```

```
In [35]: 30 > int('2000')
```

```
Out[35]: False
```

- Each type supports different operations, functions and **methods**

```
In [36]: 'ACTG'.lower()
```

```
Out[36]: 'actg'
```

```
In [37]: [1, 2, 3].lower()
```

```
-------------------------------------------------------------------------
--
AttributeError                            Traceback (most recent call las
t)
<ipython-input-37-4e1a84c0439c> in <module>()
----> 1 [1, 2, 3].lower()

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

- Convert to number: `'2000'` and `'0.5'` and `'1e9'`

```
In [38]: float('2000')

Out[38]: 2000.0
```

```
In [39]: float('0.9')

Out[39]: 0.9
```

```
In [40]: float('1e9')

Out[40]: 1000000000.0
```

```
In [41]: float('1e-2')

Out[41]: 0.01
```

```
In [42]: int('2000')

Out[42]: 2000
```

```
In [43]: int('1.5')
```

```
-------------------------------------------------------------------------
--
ValueError                                Traceback (most recent call las
t)
<ipython-input-43-1fc18d793f3f> in <module>()
----> 1 int('1.5')

ValueError: invalid literal for int() with base 10: '1.5'
```

In [44]:

```
int('1e9')
```

```
---------------------------------------------------------------------------
--
ValueError                                Traceback (most recent call las
t)
<ipython-input-44-cb568d180cc9> in <module>()
----> 1 int('1e9')

ValueError: invalid literal for int() with base 10: '1e9'
```

- Convert to boolean: `1`, `0`, `'1'`, `'0'`, `''`, `{}`

```
In [45]: bool(1)

Out[45]: True

In [46]: bool(0)

Out[46]: False

In [47]: bool('1')

Out[47]: True

In [48]: bool('0')

Out[48]: True

In [49]: bool('')

Out[49]: False

In [50]: bool({})

Out[50]: False
```

- Python and the truth: true and false values

In [51]:

```
values = [1, 0, '', '0', '1', [], [0]]
for x in values:
    if x:
        print(repr(x), 'is true!')
    else:
        print(repr(x), 'is false!')
```

```
1 is true!
0 is false!
'' is false!
'0' is true!
'1' is true!
[] is false!
[0] is true!
```

- Converting between strings and lists

```
In [52]: list("hello")

Out[52]: ['h', 'e', 'l', 'l', 'o']

In [53]: str(['h', 'e', 'l', 'l', 'o'])

Out[53]: "['h', 'e', 'l', 'l', 'o']"

In [55]: ''.join(['h', 'e', 'l', 'l', 'o'])

Out[55]: 'hello'
```

Container types, when should you use which?

- **lists**: when order is important
- **dictionaries**: to keep track of the relation between keys and values
- **sets**: to check for membership. No order, no duplicates.

In [56]:

```python
genre_list = ["comedy", "drama", "drama", "sci-fi"]
genre_list
```

Out[56]:

```
['comedy', 'drama', 'drama', 'sci-fi']
```

In [57]:

```python
genres = set(genre_list)
genres
```

Out[57]:

```
{'comedy', 'drama', 'sci-fi'}
```

In [58]:

```python
'drama' in genres
```

Out[58]:

```
True
```

In [59]:

```python
genre_counts = {"comedy": 1, "drama": 2, "sci-fi": 1}
genre_counts
```

Out[59]:

```
{'comedy': 1, 'drama': 2, 'sci-fi': 1}
```

In [60]:

```python
movie = {"rating": 10.0, "title": "Toy Story"}
movie
```

Out[60]:

```
{'rating': 10.0, 'title': 'Toy Story'}
```

What is a function?

- A named piece of code that performs a specific task
- A relation (mapping) between inputs (arguments) and output (return value)

```
def hello_function(number):
    # print the user input
    print(number)
    number += 2
    return 2
```

TODAY

- More on functions: keyword arguments, return statement...
- Reusing code:
  - comments and documentation
  - importing modules: using libraries
- Pandas - explore your data!

Let's get back to buisness!

- Continue working on IMDb (or other unfinished exercises) ~30 minutes
- Discussion session

More on functions

Scope - global variables and local function variables

In [61]:

```
movies = ['Toy story', 'Home alone']

def some_thriller_movies():
    return ['Fargo', 'The Usual Suspects']

movies = some_thriller_movies()
print(movies)
```

```
['Fargo', 'The Usual Suspects']
```

More on functions

Scope - global variables and local function variables

In [61]:

```
movies = ['Toy story', 'Home alone']

def some_thriller_movies():
    return ['Fargo', 'The Usual Suspects']

movies = some_thriller_movies()
print(movies)
```

```
['Fargo', 'The Usual Suspects']
```

In [62]:

```
movies = ['Toy story', 'Home alone']

def change_to_drama():
    movies = ['Forrest Gump', 'Titanic']

change_to_drama()
print(movies)
```

```
['Toy story', 'Home alone']
```

More on functions

Scope - global variables and local function variables

In [61]:

```
movies = ['Toy story', 'Home alone']

def some_thriller_movies():
    return ['Fargo', 'The Usual Suspects']

movies = some_thriller_movies()
print(movies)
```

```
['Fargo', 'The Usual Suspects']
```

In [62]:

```
movies = ['Toy story', 'Home alone']

def change_to_drama():
    movies = ['Forrest Gump', 'Titanic']

change_to_drama()
print(movies)
```

```
['Toy story', 'Home alone']
```

Takeaway message: be careful with your variable names!

More on functions

Scope - global variables and local function variables

In [61]:

```
movies = ['Toy story', 'Home alone']

def some_thriller_movies():
    return ['Fargo', 'The Usual Suspects']

movies = some_thriller_movies()
print(movies)
```

```
['Fargo', 'The Usual Suspects']
```

In [62]:

```
movies = ['Toy story', 'Home alone']

def change_to_drama():
    movies = ['Forrest Gump', 'Titanic']

change_to_drama()
print(movies)
```

```
['Toy story', 'Home alone']
```

Takeaway message: be careful with your variable names!

Also, global variables are usually not a good idea

More on functions

A function that counts the number of occurences of `'C'` in the argument string.

In [63]:

```
def cytosine_count(nucleotides):
    count = 0
    for x in nucleotides:
        if x == 'c' or x == 'C':
            count += 1
    return count

count1 = cytosine_count('CATATTAC')
count2 = cytosine_count('tagtag')
print(count1, count2)
```

```
2 0
```

- Functions that `return` are easier to repurpose than those that `print` their result

```
In [64]: cytosine_count('catattac') + cytosine_count('tactactac')
```

```
Out[64]: 5
```

```
In [65]: def print_cytosine_count(nucleotides):
             count = 0
             for x in nucleotides:
                 if x == 'c' or x == 'C':
                     count += 1
             print(count)

         print_cytosine_count('CATATTAC')
         print_cytosine_count('tagtag')
```

```
2
0
```

```
In [66]: print_cytosine_count('catattac') + print_cytosine_count('tactactac')
```

```
2
3

---------------------------------------------------------------------------
--
TypeError                                 Traceback (most recent call las
t)
<ipython-input-66-8fd8c197070d> in <module>()
----> 1 print_cytosine_count('catattac') + print_cytosine_count('tactacta
c')

TypeError: unsupported operand type(s) for +: 'NoneType' and 'NoneType'
```

- Functions without any `return` statement returns `None`
- Use `return` for all values you might want to use later in your program

Keyword arguments

- A way to give a name explicitly to a function for clarity

```
In [68]: sorted('file', reverse=True)

Out[68]: ['l', 'i', 'f', 'e']
```

```
In [69]: attribute = 'gene_id "unknown gene"'
         attribute.split(sep=' ', maxsplit=1)

Out[69]: ['gene_id', '"unknown gene"']
```

```
In [70]: # print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
         print('x=', end='')
         print('1')

         x=1
```

Keyword arguments

- Order of keyword arguments do not matter

```
open(file, mode='r', encoding=None) # some arguments omitted
```

- These mean the same:

```
open('files/recipes.txt', 'w', encoding='utf-8')

open('files/recipes.txt', mode='w', encoding='utf-8')

open('files/recipes.txt', encoding='utf-8', mode='w')
```

Keyword arguments

- Order of keyword arguments do not matter

```
open(file, mode='r', encoding=None) # some arguments omitted
```

- These mean the same:

```
open('files/recipes.txt', 'w', encoding='utf-8')

open('files/recipes.txt', mode='w', encoding='utf-8')

open('files/recipes.txt', encoding='utf-8', mode='w')
```

- Positional arguments comes first, keyword arguments after!

Defining functions taking keyword arguments

- Just define them as usual:

In [71]:

```
def format_sentence(subject, value, end):
    return 'The ' + subject + ' is ' + value + end

print(format_sentence('lecture', 'ongoing', '.'))

print(format_sentence('lecture', 'ongoing', end='!'))

print(format_sentence(subject='lecture', value='ongoing', end='...'))
```

```
The lecture is ongoing.
The lecture is ongoing!
The lecture is ongoing...
```

In [72]:

```
print(format_sentence(subject='lecture', 'ongoing', '.'))
```

```
  File "<ipython-input-72-8916632389ec>", line 1
    print(format_sentence(subject='lecture', 'ongoing', '.'))
                                            ^
SyntaxError: positional argument follows keyword argument
```

Defining functions with default arguments

In [74]:

```
def format_sentence(subject, value, end='.'):
    return 'The ' + subject + ' is ' + value + end

print(format_sentence('lecture', 'ongoing'))

print(format_sentence('lecture', 'ongoing', end='...'))
```

```
The lecture is ongoing.
The lecture is ongoing...
```

Defining functions with optional arguments

- Convention: use the object `None`

In [75]:

```
def format_sentence(subject, value, end='.', second_value=None):
    if second_value is None:
        return 'The ' + subject + ' is ' + value + end
    else:
        return 'The ' + subject + ' is ' + value + ' and ' + second_value + end

print(format_sentence('lecture', 'ongoing'))

print(format_sentence('lecture', 'ongoing',
                      second_value='self-referential', end='!'))
```

```
The lecture is ongoing.
The lecture is ongoing and self-referential!
```

Small detour: Python's value for missing values: `None`

- Default value for optional arguments
- Implicit return value of functions without a `return`

```
In [76]: bool(None)
```

```
Out[76]: False
```

```
In [77]: None == False, None == 0
```

```
Out[77]: (False, False)
```

```
In [78]: if None:
             print('None is true')
         else:
             print('None is not true')
```

```
None is not true
```

- Python and the truth, take two

In [79]:

```
values = [None, 1, 0, '', '0', '1', [], [0]]
for x in values:
    if x is None:
        print(repr(x), 'is None')
    if not x:
        print(repr(x), 'is false')
    if x:
        print(repr(x), 'is true')
```

```
None is None
None is false
1 is true
0 is false
'' is false
'0' is true
'1' is true
[] is false
[0] is true
```

Exercise 1

- Notebook Day_4_Exercise_1 (~30 minutes)
- Extra reading:
    - https://realpython.com/python-kwargs-and-args/ (https://realpython.com/python-kwargs-and-args/)
    - https://able.bio/rhett/python-functions-and-best-practices--78aclaa (https://able.bio/rhett/python-functions-and-best-practices--78aclaa)

A short note on code structure

- functions
- modules (files)
- documentation

Why functions?

- Cleaner code
- Better defined tasks in code
- Re-usability
- Better structure

Why modules?

- Cleaner code
- Better defined tasks in code
- Re-usability
- Better structure

Why modules?

- Cleaner code
- Better defined tasks in code
- Re-usability
- Better structure

- Collect all related functions in one file
- Import a module to use its functions
- Only need to understand what the functions do, not how

Example: sys

```
import sys

sys.argv[1]
```

or

```
import pprint
pprint.pprint(a_big_dictionary)
```

Python standard modules

Check out the module index (https://docs.python.org/3.6/py-modindex.html)

How to find the right module?

How to understand it?

How to find the right module?

- look at the module index
- search PyPI (http://pypi.org)
- ask your colleagues
- search the web!

- Standard modules: no installation needed
- Other libraries: install with `pip install` or `conda install`

How to understand it?

How to understand it?

In [83]:

```
import math

help(math.acosh)
```

```
Help on built-in function acosh in module math:

acosh(...)
    acosh(x)

    Return the inverse hyperbolic cosine of x.
```

In [84]:

```
help(str)
```

```
Help on class str in module builtins:

class str(object)
 |  str(object='') -> str
 |  str(bytes_or_buffer[, encoding[, errors]]) -> str
 |
 |  Create a new string object from the given object. If encoding or
 |  errors is specified, then the object must expose a data buffer
 |  that will be decoded using the given encoding and error handler.
 |  Otherwise, returns the result of object.__str__() (if defined)
 |  or repr(object).
 |  encoding defaults to sys.getdefaultencoding().
 |  errors defaults to 'strict'.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __format__(...)
 |      S.__format__(format_spec) -> str
 |
 |      Return a formatted version of S as described by format_spec.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |
 |  __getitem__(self, key, /)
 |      Return self[key].
 |
 |  __getnewargs__(...)
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mod__(self, value, /)
 |      Return self%value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate sign
ature.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |      Return repr(self).
 |
 |  __rmod__(self, value, /)
 |      Return value%self.
 |
 |  __rmul__(self, value, /)
 |      Return value*self.
 |
 |  __sizeof__(...)
 |      S.__sizeof__() -> size of S in memory, in bytes
 |
 |  __str__(self, /)
 |      Return str(self).
 |
 |  capitalize(...)
 |      S.capitalize() -> str
 |
 |      Return a capitalized version of S, i.e. make the first character
 |      have upper case and the rest lower case.
 |
 |  casefold(...)
 |      S.casefold() -> str
 |
 |      Return a version of S suitable for caseless comparisons.
 |
 |  center(...)
 |      S.center(width[, fillchar]) -> str
 |
 |      Return S centered in a string of length width. Padding is
 |      done using the specified fill character (default is a space)
 |
 |  count(...)
 |      S.count(sub[, start[, end]]) -> int
 |
 |      Return the number of non-overlapping occurrences of substring sub
in
 |      string S[start:end].  Optional arguments start and end are
 |      interpreted as in slice notation.
 |
 |
 |  encode(...)
 |      S.encode(encoding='utf-8', errors='strict') -> bytes
 |
 |      Encode S using the codec registered for encoding. Default encodin
g
 |      is 'utf-8'. errors may be given to set a different error
 |      handling scheme. Default is 'strict' meaning that encoding errors
raise
 |      a UnicodeEncodeError. Other possible values are 'ignore', 'replac
e' and
 |      'xmlcharrefreplace' as well as any other name registered with
 |      codecs.register_error that can handle UnicodeEncodeErrors.
 |
 |  endswith(...)
 |      S.endswith(suffix[, start[, end]]) -> bool
 |
 |      Return True if S ends with the specified suffix, False otherwise.
 |      With optional start, test S beginning at that position.
 |      With optional end, stop comparing S at that position.
 |      suffix can also be a tuple of strings to try.
 |
 |  expandtabs(...)
 |      S.expandtabs(tabsize=8) -> str
 |
 |      Return a copy of S where all tab characters are expanded using sp
aces.
 |      If tabsize is not given, a tab size of 8 characters is assumed.
 |
 |  find(...)
 |      S.find(sub[, start[, end]]) -> int
 |
 |      Return the lowest index in S where substring sub is found,
 |      such that sub is contained within S[start:end].  Optional
 |      arguments start and end are interpreted as in slice notation.
 |
 |      Return -1 on failure.
 |
 |  format(...)
 |      S.format(*args, **kwargs) -> str
 |      
 |      Return a formatted version of S, using substitutions from args an
d kwargs.
 |      The substitutions are identified by braces ('{' and '}').
 |  
 |  format_map(...)
 |      S.format_map(mapping) -> str
 |      
 |      Return a formatted version of S, using substitutions from mappin
g.
 |      The substitutions are identified by braces ('{' and '}').
 |  
 |  index(...)
 |      S.index(sub[, start[, end]]) -> int
 |      
 |      Return the lowest index in S where substring sub is found,
 |      such that sub is contained within S[start:end].  Optional
 |      arguments start and end are interpreted as in slice notation.
 |      
 |      Raises ValueError when the substring is not found.
 |  
 |  isalnum(...)
 |      S.isalnum() -> bool
 |      
 |      Return True if all characters in S are alphanumeric
 |      and there is at least one character in S, False otherwise.
 |  
 |  isalpha(...)
 |      S.isalpha() -> bool
 |      
 |      Return True if all characters in S are alphabetic
 |      and there is at least one character in S, False otherwise.
 |  
 |  isdecimal(...)
 |      S.isdecimal() -> bool
 |      
 |      Return True if there are only decimal characters in S,
 |      False otherwise.
 |
 |  isdigit(...)
 |      S.isdigit() -> bool
 |
 |      Return True if all characters in S are digits
 |      and there is at least one character in S, False otherwise.
 |
 |  isidentifier(...)
 |      S.isidentifier() -> bool
 |
 |      Return True if S is a valid identifier according
 |      to the language definition.
 |
 |      Use keyword.iskeyword() to test for reserved identifiers
 |      such as "def" and "class".
 |
 |  islower(...)
 |      S.islower() -> bool
 |
 |      Return True if all cased characters in S are lowercase and there
is
 |      at least one cased character in S, False otherwise.
 |
 |  isnumeric(...)
 |      S.isnumeric() -> bool
 |
 |      Return True if there are only numeric characters in S,
 |      False otherwise.
 |
 |  isprintable(...)
 |      S.isprintable() -> bool
 |
 |      Return True if all characters in S are considered
 |      printable in repr() or S is empty, False otherwise.
 |
 |  isspace(...)
 |      S.isspace() -> bool
 |
 |      Return True if all characters in S are whitespace
 |      and there is at least one character in S, False otherwise.
 |
 |  istitle(...)
 |      S.istitle() -> bool
 |
 |      Return True if S is a titlecased string and there is at least one
 |      character in S, i.e. upper- and titlecase characters may only
 |      follow uncased characters and lowercase characters only cased one
s.
 |      Return False otherwise.
 |
 |  isupper(...)
 |      S.isupper() -> bool
 |
 |      Return True if all cased characters in S are uppercase and there
is
 |      at least one cased character in S, False otherwise.
 |
 |  join(...)
 |      S.join(iterable) -> str
 |
 |      Return a string which is the concatenation of the strings in the
 |      iterable.  The separator between elements is S.
 |
 |  ljust(...)
 |      S.ljust(width[, fillchar]) -> str
 |
 |      Return S left-justified in a Unicode string of length width. Padd
ing is
 |      done using the specified fill character (default is a space).
 |
 |  lower(...)
 |      S.lower() -> str
 |
 |      Return a copy of the string S converted to lowercase.
 |
 |
 |  lstrip(...)
 |      S.lstrip([chars]) -> str
 |
 |      Return a copy of the string S with leading whitespace removed.
 |      If chars is given and not None, remove characters in chars instea
d.
 |
 |  partition(...)
 |      S.partition(sep) -> (head, sep, tail)
 |
 |      Search for the separator sep in S, and return the part before it,
 |      the separator itself, and the part after it.  If the separator is
not
 |      found, return S and two empty strings.
 |
 |  replace(...)
 |      S.replace(old, new[, count]) -> str
 |
 |      Return a copy of S with all occurrences of substring
 |      old replaced by new.  If the optional argument count is
 |      given, only the first count occurrences are replaced.
 |
 |  rfind(...)
 |      S.rfind(sub[, start[, end]]) -> int
 |
 |      Return the highest index in S where substring sub is found,
 |      such that sub is contained within S[start:end].  Optional
 |      arguments start and end are interpreted as in slice notation.
 |
 |      Return -1 on failure.
 |
 |  rindex(...)
 |      S.rindex(sub[, start[, end]]) -> int
 |
 |      Return the highest index in S where substring sub is found,
 |      such that sub is contained within S[start:end].  Optional
 |      arguments start and end are interpreted as in slice notation.
 |
 |
 |      Raises ValueError when the substring is not found.
 |
 |  rjust(...)
 |      S.rjust(width[, fillchar]) -> str
 |
 |      Return S right-justified in a string of length width. Padding is
 |      done using the specified fill character (default is a space).
 |
 |  rpartition(...)
 |      S.rpartition(sep) -> (head, sep, tail)
 |
 |      Search for the separator sep in S, starting at the end of S, and
return
 |      the part before it, the separator itself, and the part after it.
If the
 |      separator is not found, return two empty strings and S.
 |
 |  rsplit(...)
 |      S.rsplit(sep=None, maxsplit=-1) -> list of strings
 |
 |      Return a list of the words in S, using sep as the
 |      delimiter string, starting at the end of the string and
 |      working to the front.  If maxsplit is given, at most maxsplit
 |      splits are done. If sep is not specified, any whitespace string
 |      is a separator.
 |
 |  rstrip(...)
 |      S.rstrip([chars]) -> str
 |
 |      Return a copy of the string S with trailing whitespace removed.
 |      If chars is given and not None, remove characters in chars instea
d.
 |
 |  split(...)
 |      S.split(sep=None, maxsplit=-1) -> list of strings
 |
 |      Return a list of the words in S, using sep as the
 |      delimiter string.  If maxsplit is given, at most maxsplit
 |      splits are done. If sep is not specified or is None, any
 |      whitespace string is a separator and empty strings are
 |      removed from the result.
 |
 |  splitlines(...)
 |      S.splitlines([keepends]) -> list of strings
 |
 |      Return a list of the lines in S, breaking at line boundaries.
 |      Line breaks are not included in the resulting list unless keepend
s
 |      is given and true.
 |
 |  startswith(...)
 |      S.startswith(prefix[, start[, end]]) -> bool
 |
 |      Return True if S starts with the specified prefix, False otherwis
e.
 |      With optional start, test S beginning at that position.
 |      With optional end, stop comparing S at that position.
 |      prefix can also be a tuple of strings to try.
 |
 |  strip(...)
 |      S.strip([chars]) -> str
 |
 |      Return a copy of the string S with leading and trailing
 |      whitespace removed.
 |      If chars is given and not None, remove characters in chars instea
d.
 |
 |  swapcase(...)
 |      S.swapcase() -> str
 |
 |      Return a copy of S with uppercase characters converted to lowerca
se
 |      and vice versa.
 |
 |  title(...)
 |      S.title() -> str
 |
 |      Return a titlecased version of S, i.e. words start with title cas
e
 |      characters, all remaining cased characters have lower case.
 |
 |  translate(...)
 |      S.translate(table) -> str
 |
 |      Return a copy of the string S in which each character has been ma
pped
 |      through the given translation table. The table must implement
 |      lookup/indexing via __getitem__, for instance a dictionary or lis
t,
 |      mapping Unicode ordinals to Unicode ordinals, strings, or None. I
f
 |      this operation raises LookupError, the character is left untouche
d.
 |      Characters mapped to None are deleted.
 |
 |  upper(...)
 |      S.upper() -> str
 |
 |      Return a copy of S converted to uppercase.
 |
 |  zfill(...)
 |      S.zfill(width) -> str
 |
 |      Pad a numeric string S with zeros on the left, to fill a field
 |      of the specified width. The string S is never truncated.
 |
 |  ----------------------------------------------------------------------
-
 |  Static methods defined here:
 |
 |  maketrans(x, y=None, z=None, /)
 |      Return a translation table usable for str.translate().
 |
```

In [85]:

```
help(math.sqrt)
```

```
Help on built-in function sqrt in module math:

sqrt(...)
    sqrt(x)

    Return the square root of x.
```

```
In [86]: math.sqrt(3)

Out[86]: 1.7320508075688772
```

Importing

```
In [87]: import math

         math.sqrt(3)
```

```
Out[87]: 1.7320508075688772
```

Importing

In [87]:

```
import math

math.sqrt(3)
```

Out[87]:

```
1.7320508075688772
```

In [88]:

```
import math as m
m.sqrt(3)
```

Out[88]:

```
1.7320508075688772
```

Importing

```
In [87]: import math

         math.sqrt(3)

Out[87]: 1.7320508075688772
```

```
In [88]: import math as m
         m.sqrt(3)

Out[88]: 1.7320508075688772
```

```
In [89]: from math import sqrt
         sqrt(3)

Out[89]: 1.7320508075688772
```

Documentation and commenting your code

Remember `help()`?

Works because somebody else has documented their code!

Documentation and commenting your code

Remember `help()`?

Works because somebody else has documented their code!

In [90]:

```python
def process_file(filename, chrom, pos):
    """
    Read a vcf file, search for lines matching
    chromosome chrom and position pos.

    Print the genotypes of the matching lines.
    """
    for line in open(filename):
        if not line.startswith('#'):
            col = line.split('\t')
            if col[0] == chrom and col[1] == pos:
                print(col[9:])
```

Documentation and commenting your code

Remember `help()`?

Works because somebody else has documented their code!

In [90]:

```
def process_file(filename, chrom, pos):
    """
    Read a vcf file, search for lines matching
    chromosome chrom and position pos.

    Print the genotypes of the matching lines.
    """
    for line in open(filename):
        if not line.startswith('#'):
            col = line.split('\t')
            if col[0] == chrom and col[1] == pos:
                print(col[9:])
```

In [91]:

```
help(process_file)
```

```
Help on function process_file in module __main__:

process_file(filename, chrom, pos)
    Read a vcf file, search for lines matching
    chromosome chrom and position pos.

    Print the genotypes of the matching lines.
```

Your code may have two types of users:

- library users
- maintainers (maybe yourself!)

Your code may have two types of users:

- library users
- maintainers (maybe yourself!)

Write documentation for both of them!

- library users (docstrings):

  ```
  """
  What does this function do?
  """
  ```

- maintainers (comments):

  ```
  # implementation details
  ```

Documentation:

- At the beginning of the file

```
"""
 This module provides functions fo
r...
 """
```

- For every function

```
def make_list(x):
      """Returns a random list of length
 x."""
      ...
```

Comments:

- Wherever the code is hard to understand

```
my_list[5] += other_list[3]  # explain why you do this!
```

Read more:

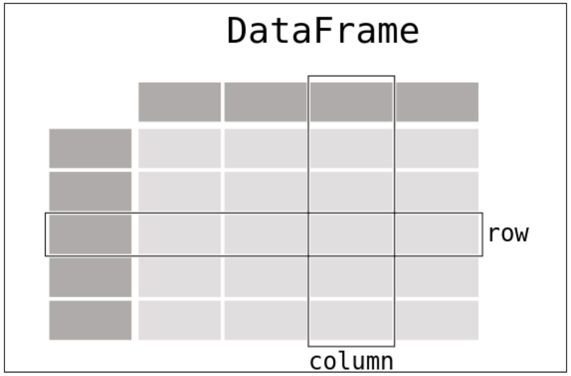
https://realpython.com/documenting-python-code/ (https://realpython.com/documenting-python-code/)

https://www.python.org/dev/peps/pep-0008/?#comments (https://www.python.org/dev/peps/pep-0008/?#comments)

Pandas!!!

Pandas

- Library for working with tabular data
- Data analysis:
  - filter
  - transform
  - aggregate
  - plot
- Main hero: the `DataFrame` type:



Creating a small DataFrame

In [142]:

```
import pandas as pd
df = pd.DataFrame({
    'age': [1,2,3,4],
    'circumference': [2,3,5,10],
    'height': [30, 35, 40, 50]
})
df
```

Out[142]:

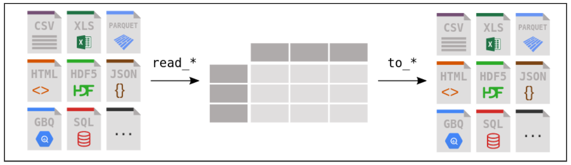
	age	circumference	height
0	1	2	30
1	2	3	35
2	3	5	40
3	4	10	50

Pandas can import data from many formats

- `pd.read_table`: tab separated values `.tsv`
- `pd.read_csv`: comma separated values `.csv`

- `pd.read_excel`: Excel spreadsheets `.xlsx`

- For a data frame `df`: `df.write_table()`, `df.write_csv()`, `df.write_excel()`



Orange tree data

In [143]:

```
!cat ../downloads/Orange_1.tsv
```

```
age	circumference	height
1	2	30
2	3	35
3	5	40
4	10	50
```

In [144]:

```
df = pd.read_table('../downloads/Orange_1.tsv')
df
```

Out[144]:

	age	circumference	height
0	1	2	30
1	2	3	35
2	3	5	40
3	4	10	50

Orange tree data

In [143]:

```
!cat ../downloads/Orange_1.tsv
```

```
age     circumference   height
1       2       30
2       3       35
3       5       40
4       10      50
```

In [144]:

```
df = pd.read_table('../downloads/Orange_1.tsv')
df
```

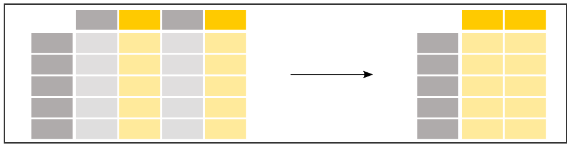
Out[144]:

	age	**circumference**	**height**
0	1	2	30
1	2	3	35
2	3	5	40
3	4	10	50

- One implict index (0, 1, 2, 3)
- Columns: `age`, `circumference`, `height`
- Rows: one per data point, identified by their index

Selecting columns from a dataframe

```
dataframe.columnname
dataframe['columnname']
```



In [145]:

```
df.columns
```

Out[145]:

```
Index(['age', 'circumference', 'height'], dtype='object')
```

In [146]:

```
df[['height', 'age']]
```

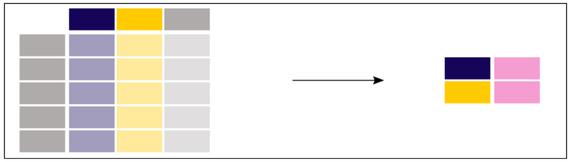
Out[146]:

	height	age
0	30	1
1	35	2
2	40	3
3	50	4

```
In [147]: df.height
```

```
Out[147]: 0    30
          1    35
          2    40
          3    50
          Name: height, dtype: int64
```

Calculating aggregated summary statistics

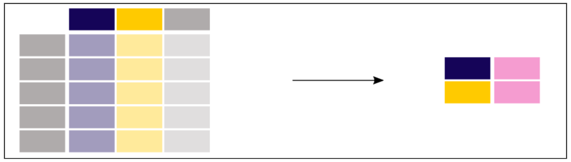


```
In [148]: df[['age', 'circumference']].describe()
```

Out[148]:

	age	circumference
count	4.000000	4.000000
mean	2.500000	5.000000
std	1.290994	3.559026
min	1.000000	2.000000
25%	1.750000	2.750000
50%	2.500000	4.000000
75%	3.250000	6.250000
max	4.000000	10.000000

Calculating aggregated summary statistics



In [148]:

```
df[['age', 'circumference']].describe()
```

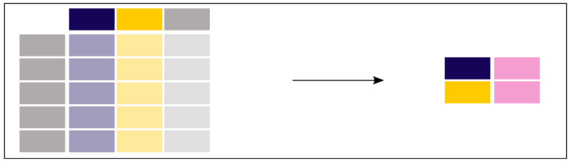
Out[148]:

	age	circumference
count	4.000000	4.000000
mean	2.500000	5.000000
std	1.290994	3.559026
min	1.000000	2.000000
25%	1.750000	2.750000
50%	2.500000	4.000000
75%	3.250000	6.250000
max	4.000000	10.000000

```
In [149]: df['age'].std()
```

```
Out[149]: 1.2909944487358056
```

Calculating aggregated summary statistics



In [148]:

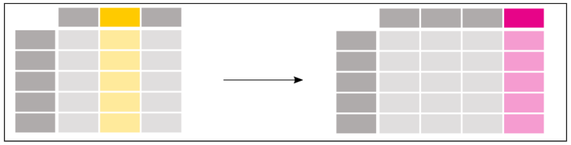
```
df[['age', 'circumference']].describe()
```

Out[148]:

	age	circumference
count	4.000000	4.000000
mean	2.500000	5.000000
std	1.290994	3.559026
min	1.000000	2.000000
25%	1.750000	2.750000
50%	2.500000	4.000000
75%	3.250000	6.250000
max	4.000000	10.000000

```
In [149]: df['age'].std()

Out[149]: 1.2909944487358056

In [150]: df['age'].max()

Out[150]: 4
```

Creating new column derived from existing column



In [151]:

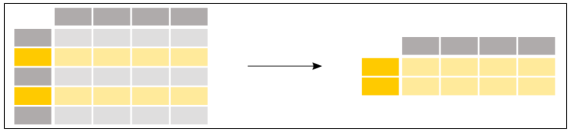
```
import math
df['radius'] = df['circumference'] / 2.0 / math.pi
df
```

Out[151]:

	age	circumference	height	radius
0	1	2	30	0.318310
1	2	3	35	0.477465
2	3	5	40	0.795775
3	4	10	50	1.591549

Selecting rows from a dataframe by index

```
dataframe.iloc[index]
dataframe.iloc[start:stop]
```



```
In [152]: df.iloc[1]
```

```
Out[152]: age                 2.000000
          circumference       3.000000
          height             35.000000
          radius              0.477465
          Name: 1, dtype: float64
```

Slightly bigger data frame of orange trees

In [153]:

```
!head -n 10 ../downloads/Orange.tsv
```

```
Tree    age     circumference
1       118     30
1       484     58
1       664     87
1       1004    115
1       1231    120
1       1372    142
1       1582    145
2       118     33
2       484     69
```

In [154]:

```
df = pd.read_table('../downloads/Orange.tsv')
df.head()
```

Out[154]:

	Tree	age	circumference
0	1	118	30
1	1	484	58
2	1	664	87
3	1	1004	115
4	1	1231	120

In [155]:

```
df.Tree.unique()
```

Out[155]:

```
array([1, 2, 3])
```

In [157]:

```
#young = df[df.age < 200]
#young
df[df.age < 1000]
```

Out[157]:

	Tree	age	circumference
0	1	118	30
1	1	484	58
2	1	664	87
7	2	118	33
8	2	484	69
9	2	664	111
14	3	118	30
15	3	484	51
16	3	664	75

Finding the maximum and then filter by it

```
df[ df.age < 200 ]
```

In [172]:

```
df.head()
```

Out[172]:

	age	circumference	height
0	1	2	30
1	2	3	35
2	3	5	40
3	4	10	50

In [159]:

```
max_c = df.circumference.max()
print(max_c)
```

```
203
```

In [160]:

```
df[(df.circumference == max_c) & (df.age > 1500)]
```

Out[160]:

	Tree	age	circumference
13	2	1582	203

Exercise

Here's a dictionary of students and their grades:

```
students = {'student': ['bob', 'sam', 'joe'], 'grade': [1, 3, 4]}
```

Use Pandas to:

- create a dataframe with this information
- get the mean value of the grades

In [161]:

```
import pandas as pd

students = {'student': ['bob', 'sam', 'joe'], 'grade': [1, 3, 4]}

stud_df = pd.DataFrame(students)


stud_df.grade.mean()
stud_df['grade'].mean()
```

Out[161]:

```
2.6666666666666665
```

Plotting

```
df.columnname.plot()
```

Plotting

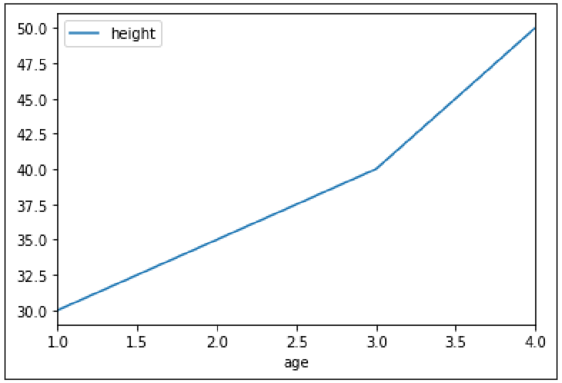
```
df.columnname.plot()
```

In [173]:

```
df = pd.read_table('../downloads/Orange_1.tsv')
df.plot(x='age', y='height')
```

Out[173]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5a91cd0710>
```



Plotting

What if no plot shows up?

```
%pylab inline   # jupyter notebooks
```

or

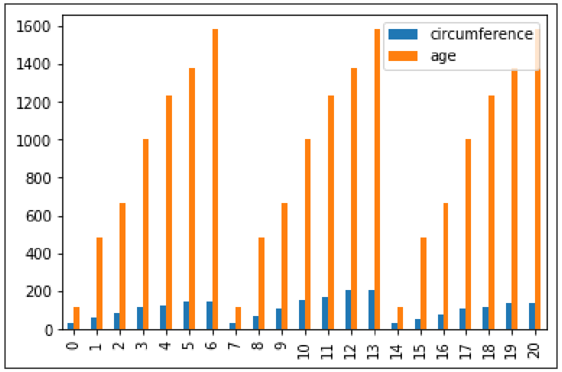
```
import matplotlib.pyplot as plt

plt.show()
```

Plotting - bars

- Plot a bar chart

```
In [166]: small_df[['circumference', 'age']].plot(kind='bar')
```

```
Out[166]: <matplotlib.axes._subplots.AxesSubplot at 0x7f5a91f09828>
```

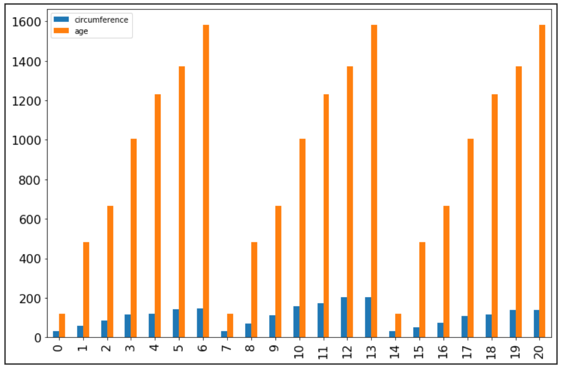


In [167]:

```
small_df[['circumference', 'age']].plot(kind='bar', figsize=(12, 8), fontsize=16)
```

Out[167]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5a91e0f978>
```

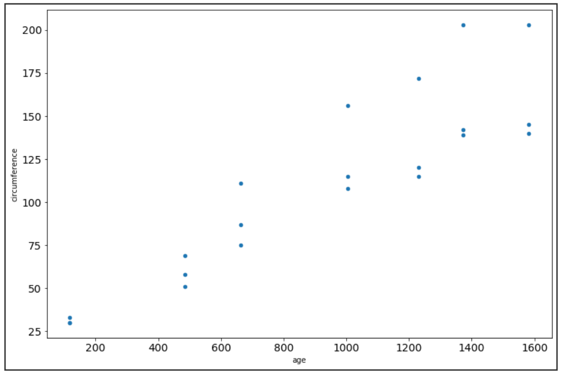


Scatterplot

```
df.plot(kind="scatter", x="column_name", y="other_column_name")
```

```
In [168]: small_df.plot(kind="scatter", x='age', y='circumference',
                  figsize=(12, 8), fontsize=14)
```

```
Out[168]: <matplotlib.axes._subplots.AxesSubplot at 0x7f5a91d0ca58>
```



Line plot

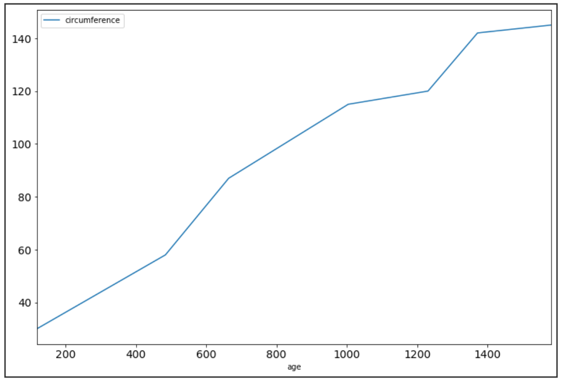
```
dataframe.plot(kind="line", x=..., y=...)
```

In [177]:

```
tree1 = small_df[small_df['Tree'] == 1]
tree1.plot(kind="line", x='age', y='circumference',
           fontsize=14, figsize=(12,8))
```

Out[177]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5a91c52a58>
```



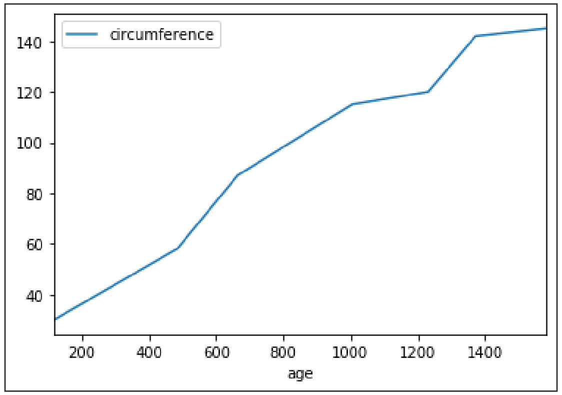
Multiple graphs - grouping

```
In [179]: small_df.groupby('Tree')
```

```
Out[179]: <pandas.core.groupby.generic.DataFrameGroupBy object at 0x7f5a91c37208>
```

```
In [181]: small_df.groupby('Tree').plot(kind="line", x='age', y='circumference')
```

```
Out[181]: Tree
          1    AxesSubplot(0.125,0.125;0.775x0.755)
          2    AxesSubplot(0.125,0.125;0.775x0.755)
          3    AxesSubplot(0.125,0.125;0.775x0.755)
          dtype: object
```



Exercise 2

- **Easy**:
    - Explore the `Orange_1.tsv`
- **Medium/hard**:
    - Use Pandas to read IMDB
    - Explore it by making graphs
- **Extra exercises**:
    - Read the pandas documentation :)
    - Start exploring your own data