Introduction to

with Application to Bioinformatics

- Day 4

TODAY

- Loops and functions, code structure
- Pandas - explore your data!

Review

- In what ways does the type of an object matter? Explain the output of:

In [2]:
```
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 [3]:
```
print(sorted([ 2000,   30,   100 ]))
print(sorted(['2000', '30', '100']))
# Hint: is `'30' > '2000'`?
```

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

- How can you convert an object to a different type?
    - Convert to number: `'2000'` and `'0.5'` and `'1e9'`
    - Convert to boolean: `1`, `0`, `'1'`, `'0'`, `''`, `{}`
- We have seen these container types: **lists**, **sets**, **dictionaries**. What is their difference and when should you use which?
- What is a function? Write a function that counts the number of occurences of `'C'` in the argument string.

In what ways does the type of an object matter?

In [4]:
```
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 [5]:
```
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 [6]:
```
30 > 2000
```

Out[6]:
```
False
```

In [7]:
```
'30' > '2000'
```

Out[7]:
```
True
```

In [8]:
```
30 > '2000'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-8-4dd71e7d1297> in <module>
----> 1 30 > '2000'

TypeError: '>' not supported between instances of 'int' and 'str'
```

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

```
In [9]: 'ACTG'.lower()

Out[9]: 'actg'

In [10]: [1, 2, 3].lower()

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-10-4e1a84c0439c> in <module>
----> 1 [1, 2, 3].lower()

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

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

In [11]:
```
int('2000')
```

Out[11]:
```
2000
```

In [12]:
```
int('0.5')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-12-6d0b04c882d1> in <module>
----> 1 int('0.5')

ValueError: invalid literal for int() with base 10: '0.5'
```

In [13]:
```
int('1e9')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-13-cb568d180cc9> in <module>
----> 1 int('1e9')

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

In [14]:
```
float('2000')
```

Out[14]:
```
2000.0
```

In [15]:
```
float('0.5')
```

Out[15]:
```
0.5
```

In [16]:
```
float('1e9')
```

Out[16]:
```
1000000000.0
```

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

In [17]:
```
bool(1)
```

Out[17]: True

In [18]:
```
bool(0)
```

Out[18]: False

In [19]:
```
bool('1')
```

Out[19]: True

In [20]:
```
bool('0')
```

Out[20]: True

In [21]:
```
bool('')
```

Out[21]: False

In [22]:
```
bool({})
```

Out[22]: False

- Python and the truth: true and false values

In [23]:
```
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 [24]:
```
list("hello")
```

Out[24]:
```
['h', 'e', 'l', 'l', 'o']
```

In [25]:
```
str(['h', 'e', 'l', 'l', 'o'])
```

Out[25]:
```
"['h', 'e', 'l', 'l', 'o']"
```

In [26]:
```
''.join(['h', 'e', 'l', 'l', 'o'])
```

Out[26]:
```
'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 [27]:
```
genre_list = ["comedy", "drama", "drama", "sci-fi"]
genre_list
```

Out[27]:
```
['comedy', 'drama', 'drama', 'sci-fi']
```

In [28]:
```
genres = set(genre_list)
genres
```

Out[28]:
```
{'comedy', 'drama', 'sci-fi'}
```

In [29]:
```
genre_counts = {"comedy": 1, "drama": 2, "sci-fi": 1}
genre_counts
```

Out[29]:
```
{'comedy': 1, 'drama': 2, 'sci-fi': 1}
```

In [30]:
```
movie = {"rating": 10.0, "title": "Toy Story"}
movie
```

Out[30]:
```
{'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)
- Write a function that counts the number of occurences of `'C'` in the argument string.

- Function for counting the number of occurences of `'C'`

In [31]:
```
def cytosine_count(nucleotides):
    count = 0
    for x in nucleotides:
        if x == 'c' or x == 'C':
            count += 1
    return count
```

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

In [32]:
```
cytosine_count('catattac') + cytosine_count('tactactac')
```

Out[32]:
```
5
```

In [33]:
```
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('tactactac')
```

```
2
3

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-33-5bbd47c30b94> in <module>
      6     print(count)
      7
----> 8 print_cytosine_count('catattac') + print_cytosine_count('tactactac')

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

- Objects and references to objects

In [34]:
```
list_A = ['red', 'green']
list_B = ['red', 'green']
list_B.append('blue')
print(list_A, list_B)
```

```
['red', 'green'] ['red', 'green', 'blue']
```

In [35]:
```
list_A = ['red', 'green']
list_B = list_A
list_B.append('blue')
print(list_A, list_B)
```

```
['red', 'green', 'blue'] ['red', 'green', 'blue']
```

In [36]:
```
list_A = ['red', 'green']
list_B = list_A
list_A = []
print(list_A, list_B)
```

```
[] ['red', 'green']
```

- Objects and references to objects, cont.

In [37]:
```
list_A = ['red', 'green']
lists = {'A': list_A, 'B': list_A}
print(lists)
lists['B'].append('blue')
print(lists)
```

```
{'A': ['red', 'green'], 'B': ['red', 'green']}
{'A': ['red', 'green', 'blue'], 'B': ['red', 'green', 'blue']}
```

In [38]:
```
list_A = ['red', 'green']
lists = {'A': list_A, 'B': list_A}
print(lists)
lists['B'] = lists['B'] + ['yellow']
print(lists)
```

```
{'A': ['red', 'green'], 'B': ['red', 'green']}
{'A': ['red', 'green'], 'B': ['red', 'green', 'yellow']}
```

Scope: global variables and local function variables

In [39]:
```
movies = ['Toy story', 'Home alone']
```

In [40]:
```
def change_to_thriller():
    movies = ['Fargo', 'The Usual Suspects']

change_to_thriller()
print(movies)
```

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

In [41]:
```
def change_to_drama(movies):
    movies = ['Forrest Gump', 'Titanic']

change_to_drama(movies)
print(movies)
```

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

In [42]:
```
def change_to_scifi(movies):
    movies.clear()
    movies += ['Terminator II', 'The Matrix']

change_to_scifi(movies)
print(movies)
```

```
['Terminator II', 'The Matrix']
```

Keyword arguments

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

In [43]:
```
sorted(list('file'), reverse=True)
```

Out[43]:
```
['l', 'i', 'f', 'e']
```

In [44]:
```
attribute = 'gene_id "unknown gene"'
attribute.split(sep=' ', maxsplit=1)
```

Out[44]:
```
['gene_id', '"unknown gene"']
```

In [45]:
```
# 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')
```

Defining functions taking keyword arguments

- Just define them as usual:

In [46]:
```
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 [47]:
```
print(format_sentence(subject='lecture', 'ongoing', '.'))
```

```
  File "<ipython-input-47-8916632389ec>", line 1
    print(format_sentence(subject='lecture', 'ongoing', '.'))
                                             ^
SyntaxError: positional argument follows keyword argument
```

- Positional arguments comes first, keyword arguments after!

Defining functions with default arguments

In [48]:
```
def format_sentence(subject, value, end='.'):
    return 'The ' + subject + ' is ' + value + end

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

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

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

Defining functions with optional arguments

- Convention: use the object `None`

In [49]:
```
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`
- Something to initialize variable with no value yet
- Argument to a function indicating use the default value

In [50]:
```
bool(None)
```

Out[50]:
```
False
```

In [51]:
```
None == False, None == 0
```

Out[51]:
```
(False, False)
```

Comparing None

- To differentiate None to the other false values such as `0`, `False` and `''` use `is None`:

In [52]:
```
counts = {'drama': 2, 'romance': 0}

counts.get('romance'), counts.get('thriller')
```

Out[52]:
```
(0, None)
```

In [53]:
```
counts.get('romance') is None
```

Out[53]:
```
False
```

In [54]:
```
counts.get('thriller') is None
```

Out[54]:
```
True
```

- Python and the truth, take two

In [55]:
```
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
```

Controlling loops - `break`

```
for x in lines_in_a_big_file:
    if x.startswith('>'):  # this is the only line I want!
        do_something(x)
```

...waste of time!

```
for x in lines_in_a_big_file:
    if x.startswith('>'):  # this is the only line I want!
        do_something(x)
        break  # break the loop
```

break

```
for line in file:
    if line.startswith('#'):
        break
    do_something(line)


print("I am done")
```

Controlling loops - `continue`

```
for x in lines_in_a_big_file:
    if x.startswith('>'):  # irrelevant line
        # just skip this! don't do anything
    do_something(x)
```

```
for x in lines_in_a_big_file:
    if x.startswith('>'):  # irrelevant line
        continue  # go on to the next iteration
    do_something(x)
```

```
for x in lines_in_a_big_file:
    if not x.startswith('>'):  # not irrelevant!
        do_something(x)
```

continue

```
for line in file:
    if line.startswith('#'):
        continue
    do_something(line)


print("I am done")
```

Another control statement: pass - the placeholder

In [56]:
```
def a_function():
    # I have not implemented this just yet
```

```
  File "<ipython-input-56-a7f30ec71867>", line 2
    # I have not implemented this just yet
                                           ^
SyntaxError: unexpected EOF while parsing
```

In [57]:
```
def a_function():
    # I have not implemented this just yet
    pass

a_function()
```

Exercise 1

- Notebook Day_4_Exercise_1 (~30 minutes)

A short note on code structure

- functions
- modules (files)
- documentation

Remember?

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

- 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 imdb_parser as imdb
imdb.parse('250.imdb')
```

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!

How to understand it?

In [58]:

```
import math

help(math)
```

```
Help on module math:

NAME
    math

MODULE REFERENCE
    https://docs.python.org/3.8/library/math

    The following documentation is automatically generated from the Python
    source files.  It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at the
    location listed above.

DESCRIPTION
    This module provides access to the mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of x.

    asinh(x, /)
        Return the inverse hyperbolic sine of x.

    atan(x, /)
        Return the arc tangent (measured in radians) of x.
```

```
    atan2(y, x, /)
        Return the arc tangent (measured in radians) of y/x.

        Unlike atan(y/x), the signs of both x and y are considered.

    atanh(x, /)
        Return the inverse hyperbolic tangent of x.

    ceil(x, /)
        Return the ceiling of x as an Integral.

        This is the smallest integer >= x.

    comb(n, k, /)
        Number of ways to choose k items from n items without repetition and wi
thout order.

        Evaluates to n! / (k! * (n - k)!) when k <= n and evaluates
        to zero when k > n.

        Also called the binomial coefficient because it is equivalent
        to the coefficient of k-th term in polynomial expansion of the
        expression (1 + x)**n.

        Raises TypeError if either of the arguments are not integers.
        Raises ValueError if either of the arguments are negative.

    copysign(x, y, /)
        Return a float with the magnitude (absolute value) of x but the sign of
y.

        On platforms that support signed zeros, copysign(1.0, -0.0)
        returns -1.0.

    cos(x, /)
        Return the cosine of x (measured in radians).

    cosh(x, /)
```

```
        Return the hyperbolic cosine of x.

    degrees(x, /)
        Convert angle x from radians to degrees.

    dist(p, q, /)
        Return the Euclidean distance between two points p and q.

        The points should be specified as sequences (or iterables) of
        coordinates.  Both inputs must have the same dimension.

        Roughly equivalent to:
            sqrt(sum((px - qx) ** 2.0 for px, qx in zip(p, q)))

    erf(x, /)
        Error function at x.

    erfc(x, /)
        Complementary error function at x.

    exp(x, /)
        Return e raised to the power of x.

    expm1(x, /)
        Return exp(x)-1.

        This function avoids the loss of precision involved in the direct evalu
ation of exp(x)-1 for small x.

    fabs(x, /)
        Return the absolute value of the float x.

    factorial(x, /)
        Find x!.

        Raise a ValueError if x is negative or non-integral.

    floor(x, /)
```

```
    Return the floor of x as an Integral.

    This is the largest integer <= x.

fmod(x, y, /)
    Return fmod(x, y), according to platform C.

    x % y may differ.

frexp(x, /)
    Return the mantissa and exponent of x, as pair (m, e).

    m is a float and e is an int, such that x = m * 2.**e.
    If x is 0, m and e are both 0.  Else 0.5 <= abs(m) < 1.0.

fsum(seq, /)
    Return an accurate floating point sum of values in the iterable seq.

    Assumes IEEE-754 floating point arithmetic.

gamma(x, /)
    Gamma function at x.

gcd(x, y, /)
    greatest common divisor of x and y

hypot(...)
    hypot(*coordinates) -> value

    Multidimensional Euclidean distance from the origin to a point.

    Roughly equivalent to:
        sqrt(sum(x**2 for x in coordinates))

    For a two dimensional point (x, y), gives the hypotenuse
    using the Pythagorean theorem:  sqrt(x*x + y*y).

    For example, the hypotenuse of a 3/4/5 right triangle is:
```

```
          >>> hypot(3.0, 4.0)
          5.0

    isclose(a, b, *, rel_tol=1e-09, abs_tol=0.0)
        Determine whether two floating point numbers are close in value.

          rel_tol
            maximum difference for being considered "close", relative to the
            magnitude of the input values
          abs_tol
            maximum difference for being considered "close", regardless of the
            magnitude of the input values

        Return True if a is close in value to b, and False otherwise.

        For the values to be considered close, the difference between them
        must be smaller than at least one of the tolerances.

        -inf, inf and NaN behave similarly to the IEEE 754 Standard.  That
        is, NaN is not close to anything, even itself.  inf and -inf are
        only close to themselves.

    isfinite(x, /)
        Return True if x is neither an infinity nor a NaN, and False otherwise.

    isinf(x, /)
        Return True if x is a positive or negative infinity, and False otherwis
e.

    isnan(x, /)
        Return True if x is a NaN (not a number), and False otherwise.

    isqrt(n, /)
        Return the integer part of the square root of the input.

    ldexp(x, i, /)
        Return x * (2**i).
```

```
        This is essentially the inverse of frexp().

    lgamma(x, /)
        Natural logarithm of absolute value of Gamma function at x.

    log(...)
        log(x, [base=math.e])
        Return the logarithm of x to the given base.

        If the base not specified, returns the natural logarithm (base e) of x.

    log10(x, /)
        Return the base 10 logarithm of x.

    log1p(x, /)
        Return the natural logarithm of 1+x (base e).

        The result is computed in a way which is accurate for x near zero.

    log2(x, /)
        Return the base 2 logarithm of x.

    modf(x, /)
        Return the fractional and integer parts of x.

        Both results carry the sign of x and are floats.

    perm(n, k=None, /)
        Number of ways to choose k items from n items without repetition and wi
th order.

        Evaluates to n! / (n - k)! when k <= n and evaluates
        to zero when k > n.

        If k is not specified or is None, then k defaults to n
        and the function returns n!.
```

```
    Raises TypeError if either of the arguments are not integers.
    Raises ValueError if either of the arguments are negative.

pow(x, y, /)
    Return x**y (x to the power of y).

prod(iterable, /, *, start=1)
    Calculate the product of all the elements in the input iterable.

    The default start value for the product is 1.

    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.

radians(x, /)
    Convert angle x from degrees to radians.

remainder(x, y, /)
    Difference between x and the closest integer multiple of y.

    Return x - n*y where n*y is the closest integer multiple of y.
    In the case where x is exactly halfway between two multiples of
    y, the nearest even value of n is used. The result is always exact.

sin(x, /)
    Return the sine of x (measured in radians).

sinh(x, /)
    Return the hyperbolic sine of x.

sqrt(x, /)
    Return the square root of x.

tan(x, /)
    Return the tangent of x (measured in radians).

tanh(x, /)
```

In [59]:
```
dir(math)
```

Out[59]:
```
['__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'acos',
 'acosh',
 'asin',
 'asinh',
 'atan',
 'atan2',
 'atanh',
 'ceil',
 'comb',
 'copysign',
 'cos',
 'cosh',
 'degrees',
 'dist',
 'e',
 'erf',
 'erfc',
 'exp',
 'expm1',
 'fabs',
 'factorial',
 'floor',
 'fmod',
 'frexp',
 'fsum',
 'gamma',
 'gcd',
 'hypot',
 'inf',
```

```
'isclose',
'isfinite',
'isinf',
'isnan',
'isqrt',
'ldexp',
'lgamma',
'log',
'log10',
'log1p',
'log2',
'modf',
'nan',
'perm',
'pi',
'pow',
'prod',
'radians',
'remainder',
'sin',
'sinh',
'sqrt',
'tan',
'tanh',
'tau',
'trunc']
```

In [60]:
```
help(math.sqrt)
```

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

sqrt(x, /)
    Return the square root of x.
```

In [61]:
```
math.sqrt(3)
```

Out[61]: 1.7320508075688772

Importing

In [62]:
```
import math

math.sqrt(3)
```

Out[62]: 1.7320508075688772

In [63]:
```
import math as m
m.sqrt(3)
```

Out[63]: 1.7320508075688772

In [64]:
```
from math import sqrt
sqrt(3)
```

Out[64]: 1.7320508075688772

Documentation and commenting your code

Remember `help()`?

Works because somebody else has documented their code!

In [65]:
```
def process_file(filename, chrom, pos):
    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 [66]:
```
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 [67]:
```
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!)

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."""
      pass
```

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)

Formatting

In [69]:
```
title = 'Toy Story'
rating = 10
print('The result is: ' + title + ' with rating: ' + str(rating))
```

```
The result is: Toy Story with rating: 10
```

In [70]:
```
# f-strings (since python 3.6)
print(f'The result is: {title} with rating: {rating}')
```

```
The result is: Toy Story with rating: 10
```

In [71]:
```
# format method
print('The result is: {} with rating: {}'.format(title, rating))
```

```
The result is: Toy Story with rating: 10
```

In [72]:
```
# the ancient way (python 2)
print('The result is: %s with rating: %s' % (title, rating))
```

```
The result is: Toy Story with rating: 10
```

Learn more from the Python docs: https://docs.python.org/3.4/library/string.html#format-string-syntax (https://docs.python.org/3.4/library/string.html#format-string-syntax)

Exercise 2

```
pick_movie(year=1996, rating_min=8.5)
The Bandit
pick_movie(rating_max=8.0, genre="Mystery")
Twelve Monkeys
```

- Notebook Day_4_Exercise_2

Pandas

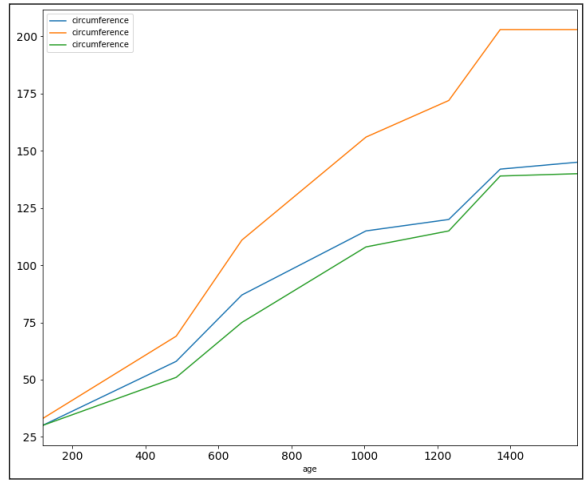
Library for working with tabular data

- comma separated (csv)
- tab separated (tsv)
- ...

Data analysis, graph plotting...

Pandas

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



Pandas - a short overview

In [73]:
```
import pandas as pd
```

In [74]:
```
help(pd)
```

```
Help on package pandas:

NAME
    pandas

DESCRIPTION
    pandas - a powerful data analysis and manipulation library for Python
    =====================================================================

    **pandas** is a Python package providing fast, flexible, and expressive da
ta
    structures designed to make working with "relational" or "labeled" data bo
th
    easy and intuitive. It aims to be the fundamental high-level building bloc
k for
    doing practical, **real world** data analysis in Python. Additionally, it
has
    the broader goal of becoming **the most powerful and flexible open source
data
    analysis / manipulation tool available in any language**. It is already we
ll on
    its way toward this goal.

    Main Features
    -------------
    Here are just a few of the things that pandas does well:

      - Easy handling of missing data in floating point as well as non-floatin
g
        point data.
      - Size mutability: columns can be inserted and deleted from DataFrame an
d
        higher dimensional objects
      - Automatic and explicit data alignment: objects can be explicitly align
ed
```

```
        to a set of labels, or the user can simply ignore the labels and let
        `Series`, `DataFrame`, etc. automatically align the data for you in
        computations.
      - Powerful, flexible group by functionality to perform split-apply-combi
ne
        operations on data sets, for both aggregating and transforming data.
      - Make it easy to convert ragged, differently-indexed data in other Pyth
on
        and NumPy data structures into DataFrame objects.
      - Intelligent label-based slicing, fancy indexing, and subsetting of lar
ge
        data sets.
      - Intuitive merging and joining data sets.
      - Flexible reshaping and pivoting of data sets.
      - Hierarchical labeling of axes (possible to have multiple labels per ti
ck).
      - Robust IO tools for loading data from flat files (CSV and delimited),
        Excel files, databases, and saving/loading data from the ultrafast HDF
5
        format.
      - Time series-specific functionality: date range generation and frequenc
y
        conversion, moving window statistics, date shifting and lagging.

PACKAGE CONTENTS
    _config (package)
    _libs (package)
    _testing
    _typing
    _version
    api (package)
    arrays (package)
    compat (package)
    conftest
    core (package)
    errors (package)
    io (package)
    plotting (package)
```

Orange tree data

- `Orange_1.tsv`:

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

In [75]:
```
tree_growth = pd.read_table('../downloads/Orange_1.tsv', index_col=0)
```

Dataframes

In [76]:
```
tree_growth
```

Out[76]:

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

- One index (in this case age)
- A bunch of colums (in this case `circumference` and `height`)
- A bunch of rows (identified by their index)

In [77]:
```
tree_growth.columns
```

Out[77]:
```
Index(['circumference', 'height'], dtype='object')
```

In [78]:
```
tree_growth.index
```

Out[78]:
```
Int64Index([1, 2, 3, 4], dtype='int64', name='age')
```

Exploring the data - picking a column

In [79]:
```
tree_growth.circumference
```

Out[79]:
```
age
1     2
2     3
3     5
4    10
Name: circumference, dtype: int64
```

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

In [80]:
```
tree_growth.height
```

Out[80]:
```
age
1    30
2    35
3    40
4    50
Name: height, dtype: int64
```

In [81]:
```
tree_growth.circumference.max()
```

Out[81]:
```
10
```

Exploring the data - picking a row

In [82]:
```
tree_growth.loc[4]
```

Out[82]:
```
circumference    10
height           50
Name: 4, dtype: int64
```

`dataframe.loc[row_name]`

Reading data

```
dataframe = pandas.read_table(filepath, index_col=N)
dataframe.columnname
dataframe.loc[row_name]
```

In [83]:
```
tree_growth = pd.read_table('../downloads/Orange_1.tsv', index_col=0)
tree_growth
```

Out[83]:

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

In [84]:
```
tree_growth.height
```

Out[84]:
```
age
1    30
2    35
3    40
4    50
Name: height, dtype: int64
```

In [85]:
```
tree_growth.loc[2]
```

Out[85]:
```
circumference     3
height           35
Name: 2, dtype: int64
```

Many trees!

- `Orange.tsv`

```
Tree    age circumfere
nce
1     118       30
1     484       58
1     664       87
1    1004      115
...
2     118       33
2     484       69
...
```

In [86]:
```
tree_growth = pd.read_table('../downloads/Orange.tsv', index_col=0)
tree_growth
```

Out[86]:

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
2	664	111
2	1004	156
2	1231	172
2	1372	203
2	1582	203
3	118	30
3	484	51
3	664	75
3	1004	108
3	1231	115
3	1372	139
3	1582	140

In [87]:
```
tree_growth.index
```

Out[87]:
```
Int64Index([1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3], dty
pe='int64', name='Tree')
```

In [88]:
```
tree_growth.columns
```

Out[88]:
```
Index(['age', 'circumference'], dtype='object')
```

In [91]:
```
tree_growth.age
```

Out[91]:
```
Tree
1     118
1     484
1     664
1    1004
1    1231
1    1372
1    1582
2     118
2     484
2     664
2    1004
2    1231
2    1372
2    1582
3     118
3     484
3     664
3    1004
3    1231
3    1372
3    1582
Name: age, dtype: int64
```

In [92]:
```
tree_growth.age.values
```

Out[92]:
```
array([ 118,  484,  664, 1004, 1231, 1372, 1582,  118,  484,  664, 1004,
       1231, 1372, 1582,  118,  484,  664, 1004, 1231, 1372, 1582])
```

In [93]:
```
tree_growth.age.unique()
```

Out[93]:
```
array([ 118,  484,  664, 1004, 1231, 1372, 1582])
```

Works like a normal list:

In [94]:
```
tree_growth.age.unique()[0]
```

Out[94]:
```
118
```

In [95]:
```
len(tree_growth.age.unique())
```

Out[95]:
```
7
```

Columns

`dataframe.columnname`

- Methods: `.max()`, `.min()`, `unique()`, `.values`, `.mean()`, `.sum()`...

Selecting parts of the table

In [96]:
```
tree_growth.circumference   # selecting a column
```

Out[96]:
```
Tree
1      30
1      58
1      87
1     115
1     120
1     142
1     145
2      33
2      69
2     111
2     156
2     172
2     203
2     203
3      30
3      51
3      75
3     108
3     115
3     139
3     140
Name: circumference, dtype: int64
```

In [97]:
```
tree_growth.loc[2]  # selecting rows with index 2
```

Out[97]:

Tree	age	circumference
2	118	33
2	484	69
2	664	111
2	1004	156
2	1231	172
2	1372	203
2	1582	203

```
# select all rows that fullfills a criteria:
tree_growth.loc[ criteria ]
```

Selecting parts of the table

Find the data points where the tree is younger than 200 years!

- Find *rows* => use `tree_growth.loc[]`
- Select these based on the value of column age => `tree_growth.age`

In [98]:
```
# The answer...
```

In [99]:
```
young = tree_growth.loc[tree_growth.age < 200]
young
```

Out[99]:

Tree	age	circumference
1	118	30
2	118	33
3	118	30

Exercises

```
tree_growth.loc[ tree_growth.age < 200 ]
```

In [100]:
```
import pandas as pd

tree_growth = pd.read_table('../downloads/Orange.tsv',index_col=0)
```

In [101]:
```
max_c = tree_growth.circumference.max()

print(max_c)
```

```
203
```

In [102]:
```
tree_growth.loc[tree_growth.circumference == max_c]
```

Out[102]:

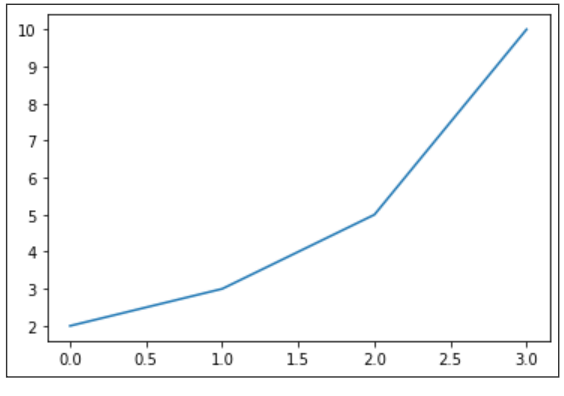
Tree	age	circumference
2	1372	203
2	1582	203

Plotting

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

In [103]:
```
orange_1 = pd.read_table('../downloads/Orange_1.tsv')
orange_1.circumference.plot()
```

Out[103]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6e4654f7c0>
```



Plotting

What if no plot shows up?

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

or

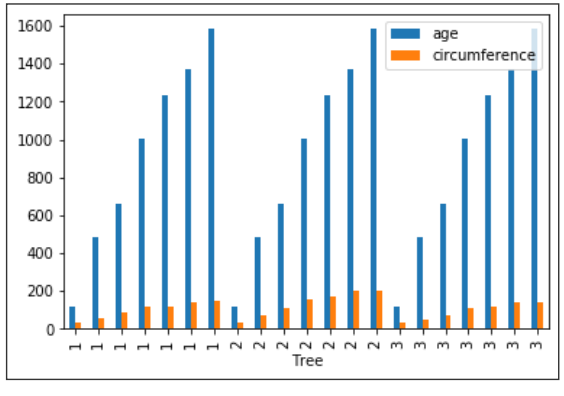
```
import matplotlib.plot as plt

plt.show()
```

Plotting - many trees

- Plot a bar chart

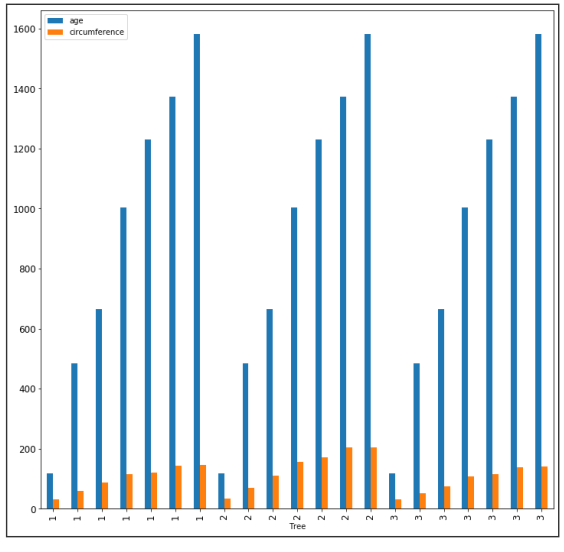
In [104]:
```
tree_growth.plot(kind='bar')
```

Out[104]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6e4424f6a0>
```



In [105]:
```
tree_growth.plot(kind='bar', figsize=(12, 12), fontsize=12)
```

Out[105]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6e43a7eeb0>



Plotting

- Plot a line graph

In [106]:
```
# Starting with tree number 1
tree1 = tree_growth.loc[1]
```

In [107]:
```
tree1
```

Out[107]:

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

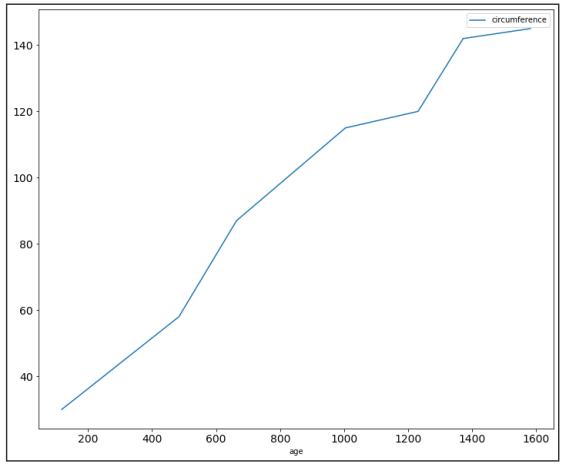
Plotting

- Plot a graph:

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

In [108]:
```
tree1.plot(x='age', y='circumference',
           fontsize=14, figsize=(12,10))
```

Out[108]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6e43a73dc0>
```



Plotting

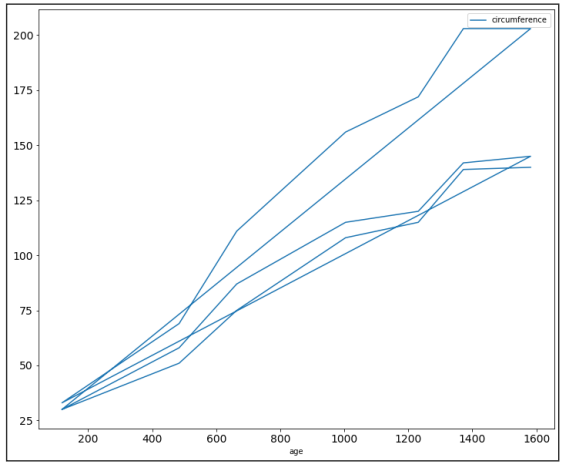
- Plot a graph:

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

Let's plot all the trees!

In [109]:
```
tree_growth.plot(kind='line', x='age', y='circumference',
                 figsize=(12, 10), fontsize=14)
```

Out[109]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6e44237be0>



:(

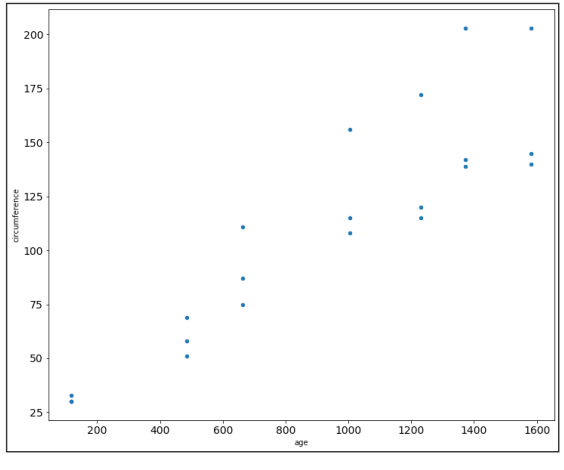
Plotting

- Plot a graph:

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

In [110]:
```
tree_growth.plot(kind='scatter', x='age', y='circumference',
                 figsize=(12, 10), fontsize=14)
```

Out[110]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6e43949a30>



Plotting

What about the lines?

- Group the table by the index (make subtrees)
- Get one board to plot all the lines
- Draw them one by one

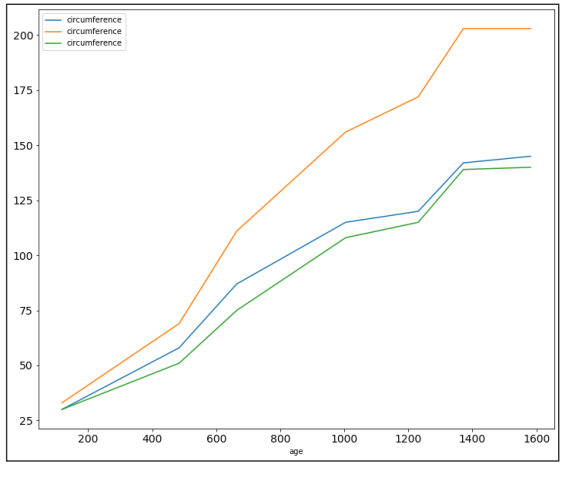
```
dataframe.groupby([what])
```

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
```

Plotting, several lines

In [111]:

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

for index, subtree in tree_growth.groupby(['Tree']):
    subtree.plot(x='age', y='circumference', kind='line',
                 ax=ax,
                 fontsize=14, figsize=(12,10))
```



Exercise 5

- Read the `Orange_1.tsv`
    - Print the height column
    - Print the data for the tree at age 2
    - Find the maximum circumference
    - What tree reached that circumference, and how old was it at that time?
- Use Pandas to read IMDB
    - Explore it by making graphs