# CS152
# Introduction to Programming (CS0) with Python

http://www.cs.colostate.edu/~cs152

Colorado State University

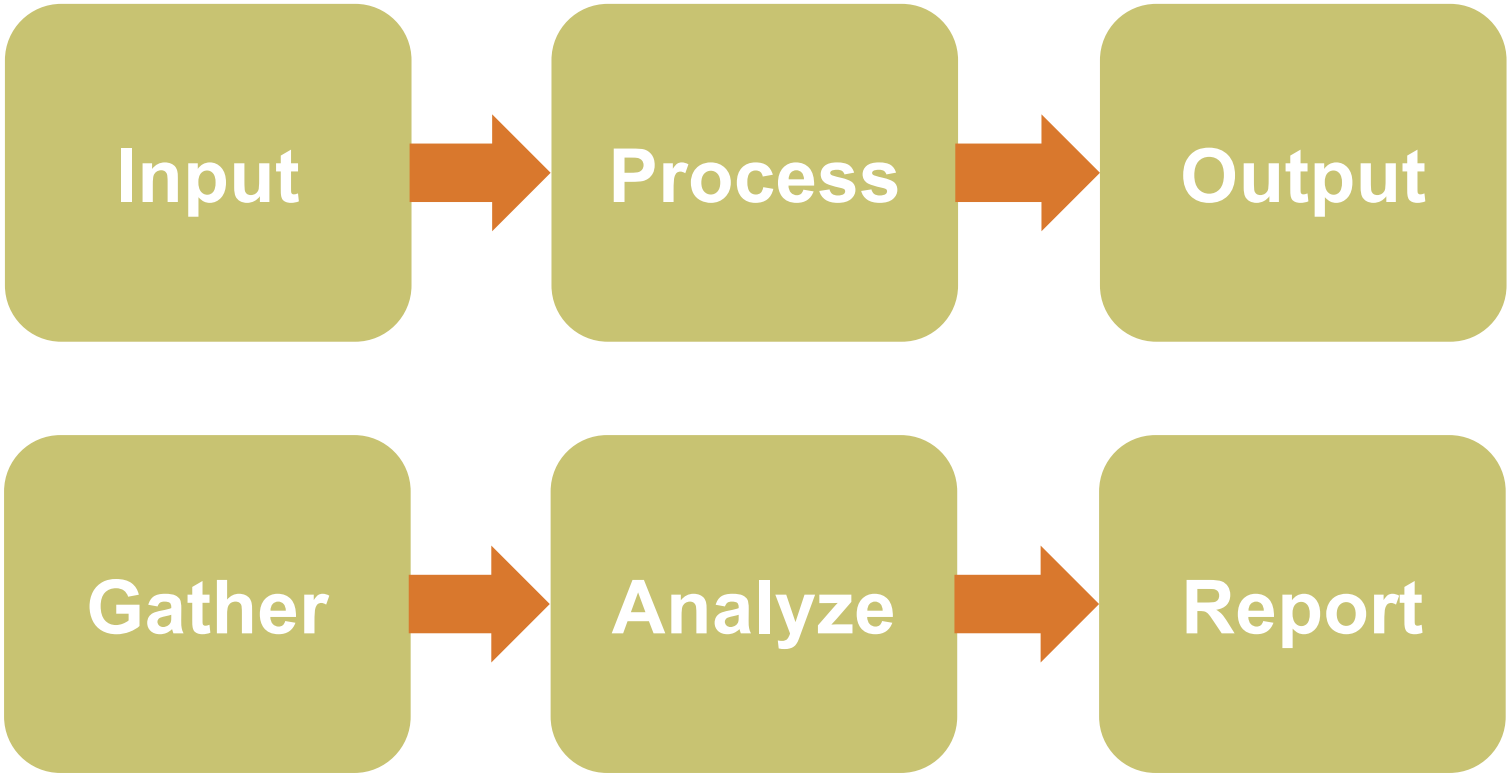
# Programming is

- Problem Solving
- Language
- Logic
- Math
- Fun!

# Programming is a bit like construction

| Construction | Programming |
| --- | --- |
| Foundation, Framing | Modules, Functions |
| Utilities<br>(Electric, Plumbing, HVAC) | Blocks<br>(condition, iteration, context) |
| Finishing<br>(walls, cabinets, trim, fixtures) | Statements, Expressions |

# Problem Solving Patterns



# Programming languages*

- Ada
- APL
- Assembly*
- Basic*
- C
- COBOL*
- Forth
- FORTRAN*
- Java*
- Lisp*
- Pascal*
- PL/I
- PL/S
- Prolog
- Python*
- R
- Scheme
- SNOBOL4
- SPL
- a few hundred more

Programming is

90% reading,

10% writing

# Aspects of programming

- language syntax and semantics
- technical writing
- engineering (problem solving)
- philosophy (logic)
- math

# Syntax

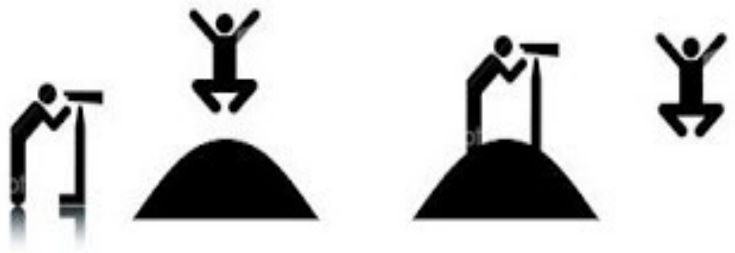
| Language | Programming |
|---|---|
| words, punctuation, numbers, emoticons | variables, operators, numbers, strings |
| phrase, clause | expressions |
| sentence | statements (assignment, print, import,…) |
| paragraph | blocks (condition, iteration, context) |
| chapter | function |
| book | module |
| library | library |

# Semantics

| Language | Programming |
| --- | --- |
| The quick brown fox<br>jumps over<br>the lazy dog. | fox(quick, brown)<br>dog(lazy)<br>jumpover(fox, dog) |
| Pack<br>my box<br>with<br>five dozen liquor jugs. | box(my)<br>jugs(five dozen, liquor)<br>packwith(box, jugs); |

# I saw the person on the hill with a telescope

**person(on the hill)**
**me(with a telescope)**
**saw(me, person)**

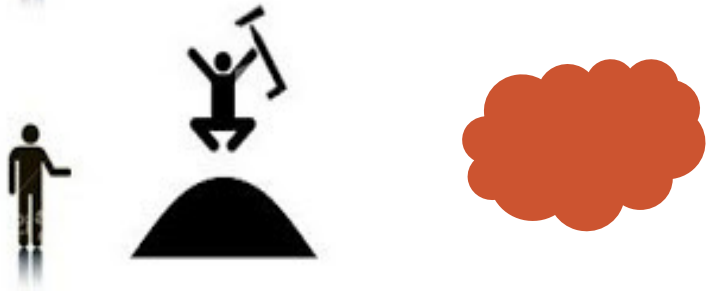


**person()**
**me(on the hill,**
**with a telescope)**
**saw(me, person)**

**hill(with a telescope)**
**person(on the hill)**
**me()**
**saw(me, person)**

**hill(with a telescope)**
**person()**
**me( on the hill)**
**saw(me, person)**

**person(on the hill,**
**with a telescope)**
**me()**
**saw(me , person)**



**person(on the hill)**
**me(with a telescope)**
**cut(me, person)**

# Learning to program requires practice!

| Mon | Tue | Wed | Thu | Fri |
|---|---|---|---|---|
| Lecture | Recitation | Lecture | Recitation | Exam |
| zyBook **P#** before class, iClicker **Q#** in class | zyLab **L#** 11:59PM | zyBook **P%** before class, iClicker **Q%** in class | zyLab **L%** 11:59PM | Exam week 3, 5, 7 |
| | zyBook **C#** 11:59PM | zyLab **A#**, 11:59PM | zyBook **C%** 11:59PM | zyLab **A%** 11:59PM |

**P# - zyBooks Participation activities**
**C# - zyBooks Challenge activities**

**L# - zyLabs Lab in chapter**
**A# - zyLabs Assignment in Chapter**

# This course is more work than expected!

- It's only a 2 credit course, but it is done in 8 weeks!
- So, it is taught at the rate of a 4 credit course.

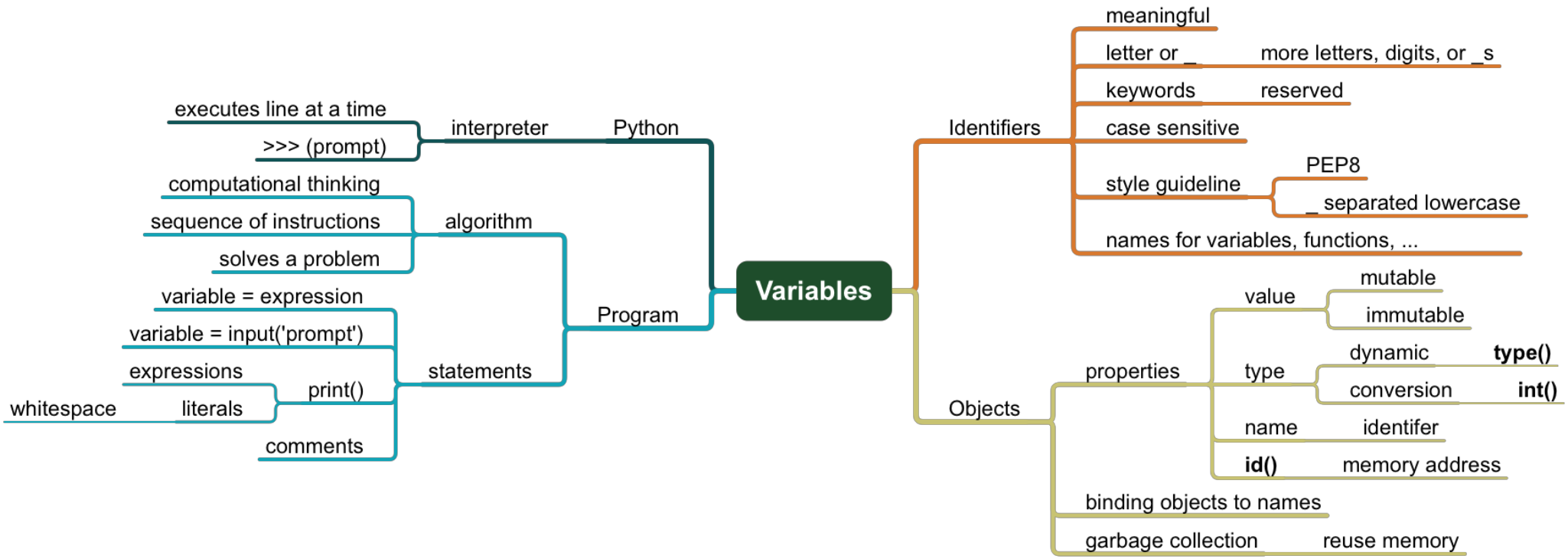
- 4 hours in class (lecture/recitation) per week
- 2 hours outside of class for each hour in class
- 8+ hours outside class per week (YMMV)
  - ~2-3 hours in zyBooks reading
  - ~3-5 hours writing programs

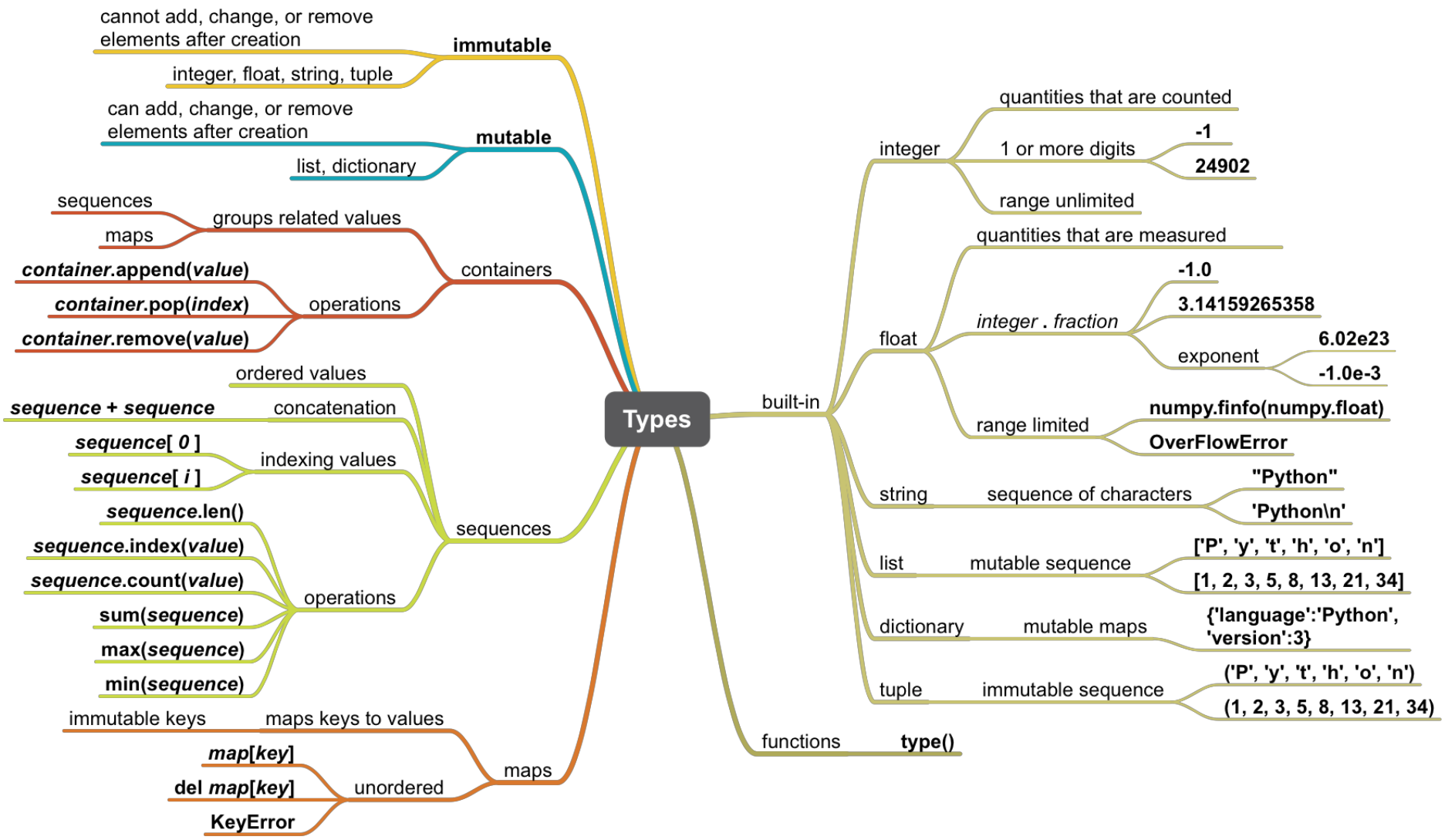
# Resources

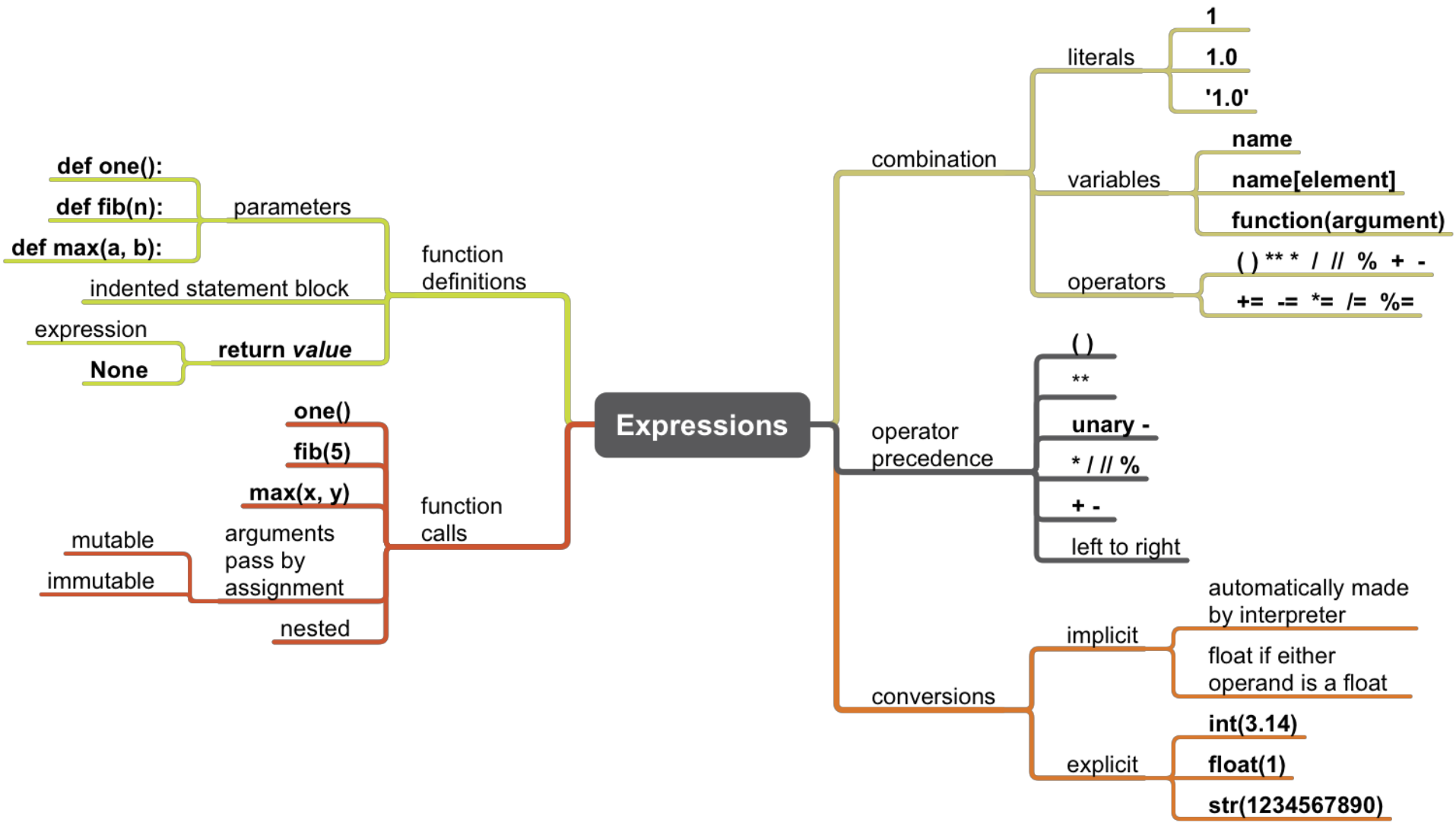
- http://www.cs.colostate.edu/~cs152
- Canvas (grades)
- zyBooks (reading, activities, labs, assignments, register)
- iClicker (lecture quizzes, register)
- PyCharm (integrated development environment)

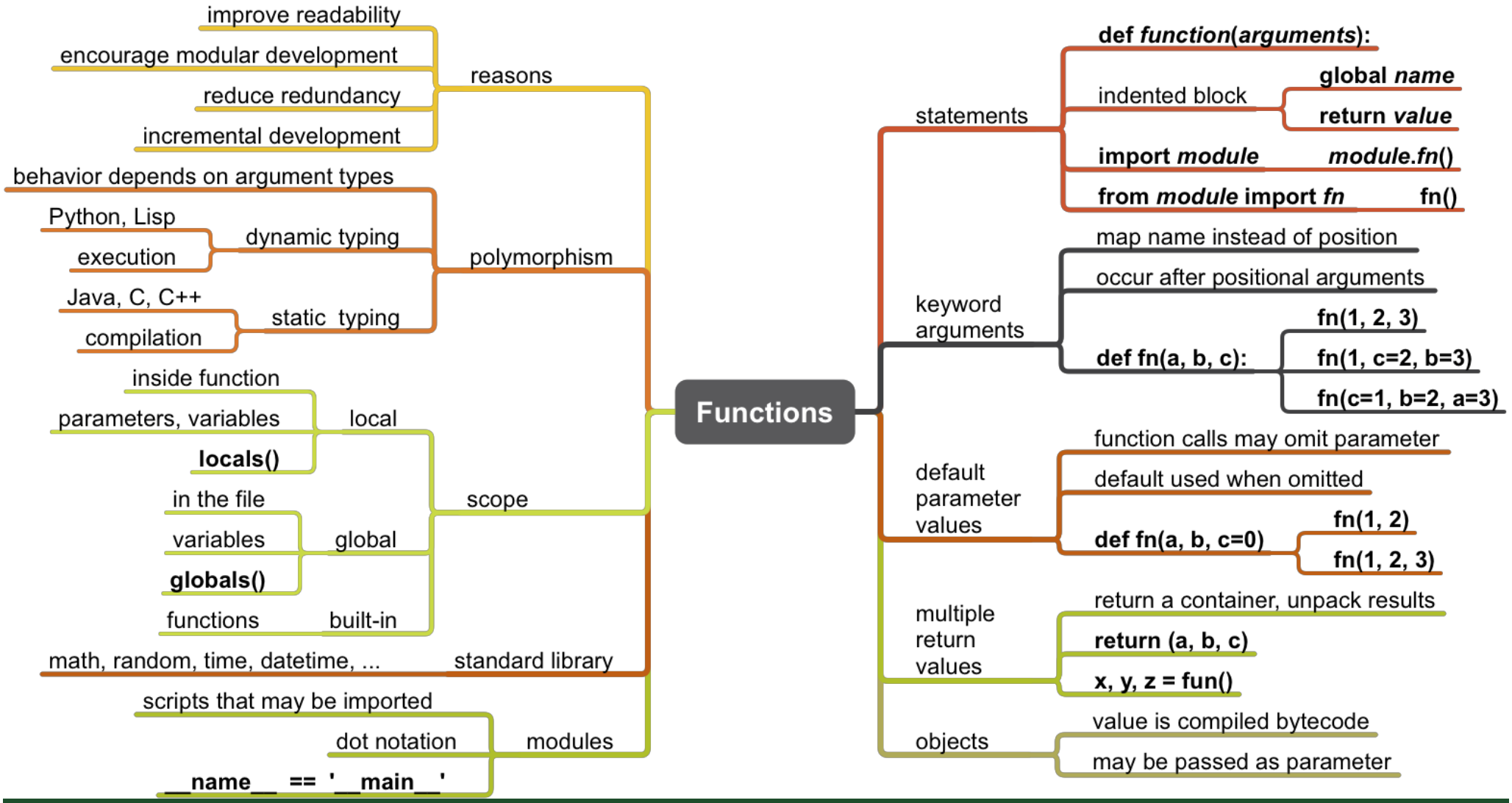
# Lecture / Study Guide

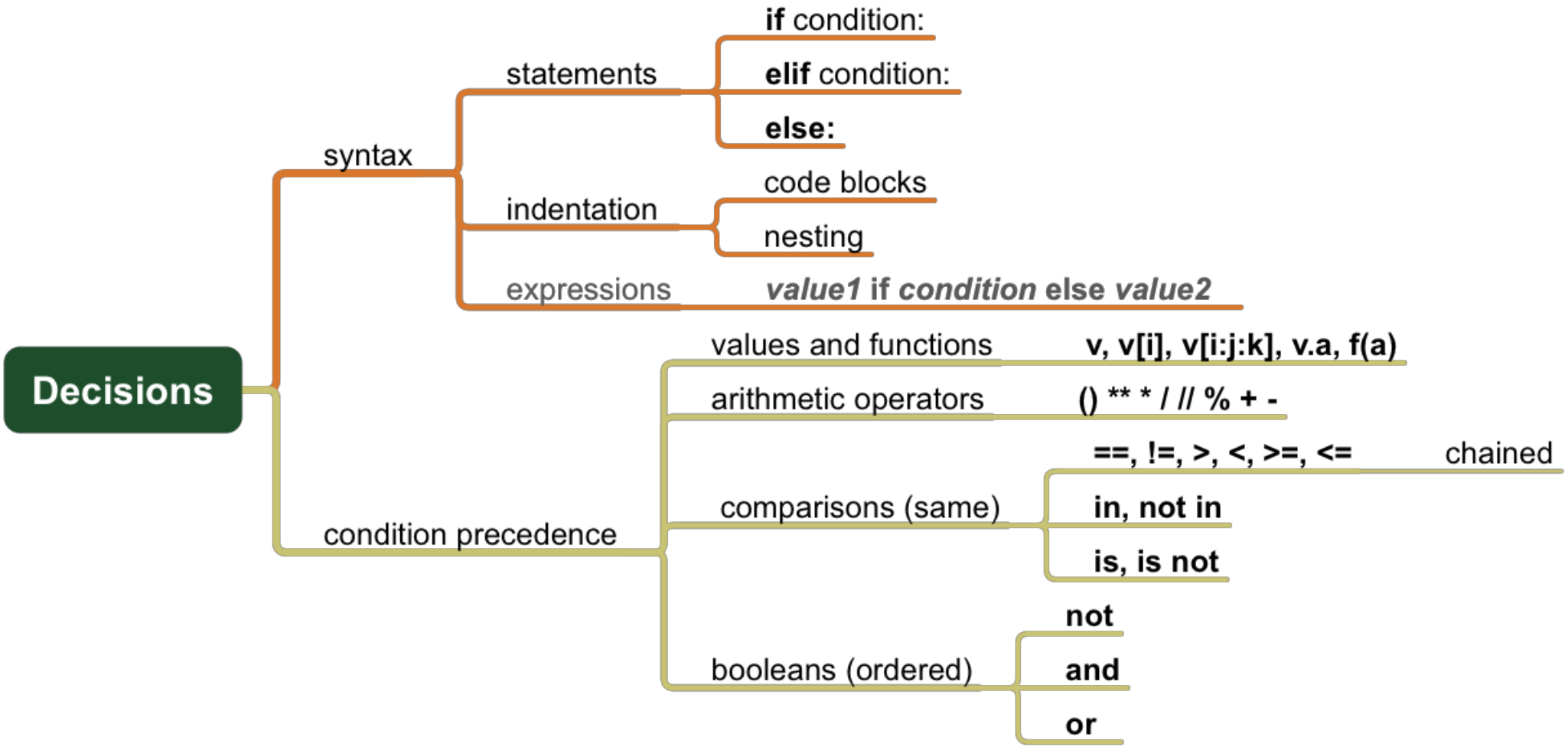
One per chapter

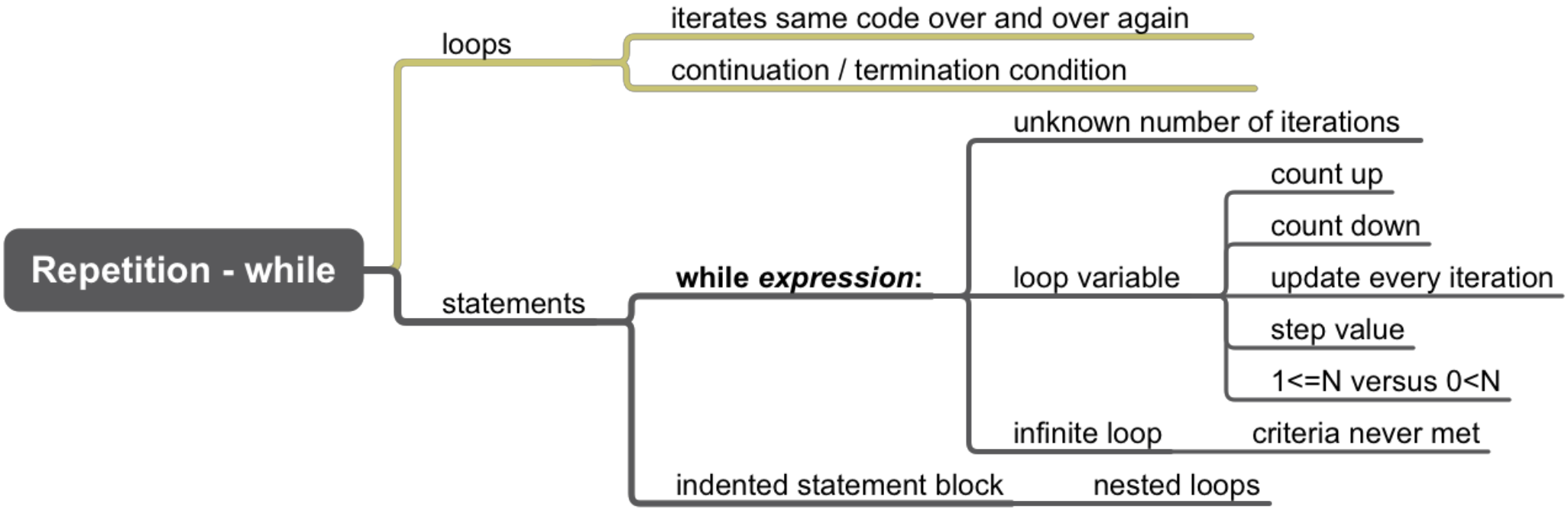

Variables
Python
interpreter
executes line at a time
>>> (prompt)
Program
algorithm
computational thinking
sequence of instructions
solves a problem
statements
variable = expression
variable = input('prompt')
expressions
print()
literals
whitespace
comments
Identifiers
meaningful
letter or _
more letters, digits, or _s
keywords
reserved
case sensitive
style guideline
PEP8
_ separated lowercase
names for variables, functions, ...
Objects
properties
value
mutable
immutable
type
dynamic
type()
conversion
int()
name
identifer
id()
memory address
binding objects to names
garbage collection
reuse memory

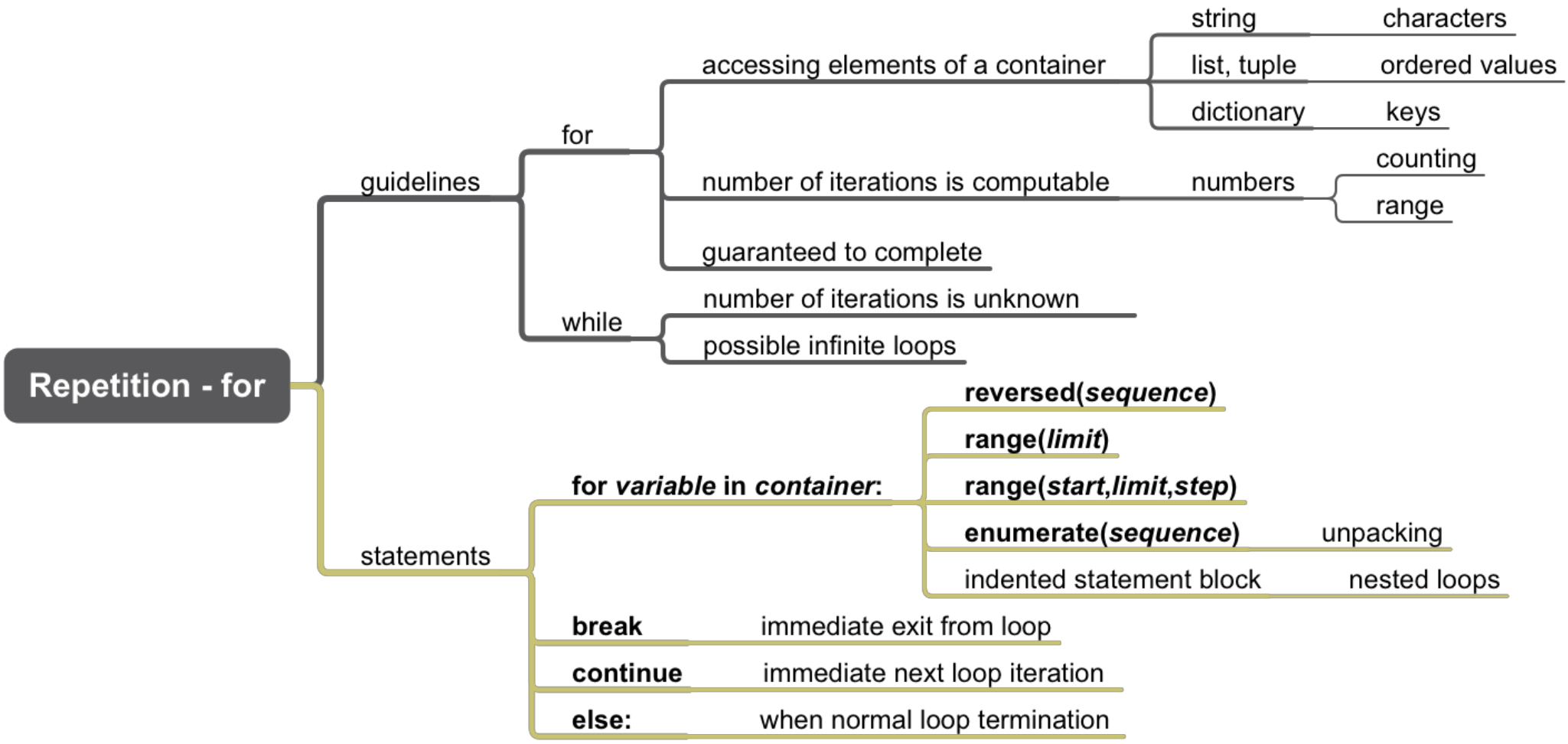

Types
built-in
integer
quantities that are counted
1 or more digits
-1
24902
range unlimited
float
quantities that are measured
integer . fraction
-1.0
3.14159265358
exponent
6.02e23
-1.0e-3
range limited
numpy.finfo(numpy.float)
OverFlowError
string
sequence of characters
"Python"
'Python\n'
list
mutable sequence
['P', 'y', 't', 'h', 'o', 'n']
[1, 2, 3, 5, 8, 13, 21, 34]
dictionary
mutable maps
{'language':'Python', 'version':3}
tuple
immutable sequence
('P', 'y', 't', 'h', 'o', 'n')
(1, 2, 3, 5, 8, 13, 21, 34)
functions
type()
immutable
cannot add, change, or remove elements after creation
integer, float, string, tuple
mutable
can add, change, or remove elements after creation
list, dictionary
containers
groups related values
sequences
maps
operations
container.append(value)
container.pop(index)
container.remove(value)
sequences
ordered values
concatenation
sequence + sequence
indexing values
sequence[ 0 ]
sequence[ i ]
operations
sequence.len()
sequence.index(value)
sequence.count(value)
sum(sequence)
max(sequence)
min(sequence)
maps
maps keys to values
immutable keys
unordered
map[key]
del map[key]
KeyError

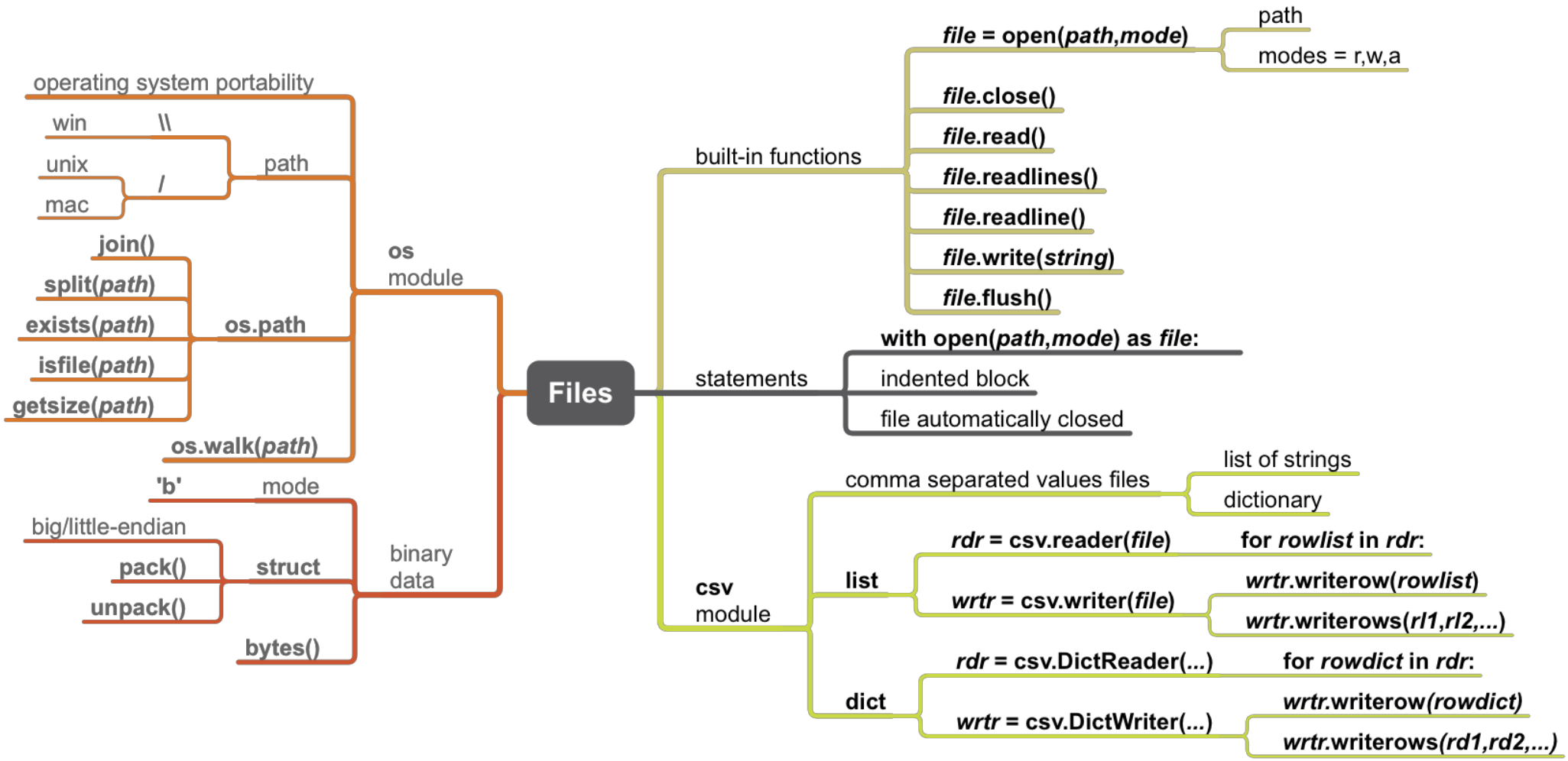

Expressions
function definitions
def one():
def fib(n):
def max(a, b):
parameters
indented statement block
expression
None
return value
function calls
one()
fib(5)
max(x, y)
arguments pass by assignment
mutable
immutable
nested
combination
literals
1
1.0
'1.0'
variables
name
name[element]
function(argument)
operators
( ) ** * / // % + -
+= -= *= /= %=
operator precedence
( )
**
unary -
* / // %
+ -
left to right
conversions
implicit
automatically made by interpreter
float if either operand is a float
explicit
int(3.14)
float(1)
str(1234567890)

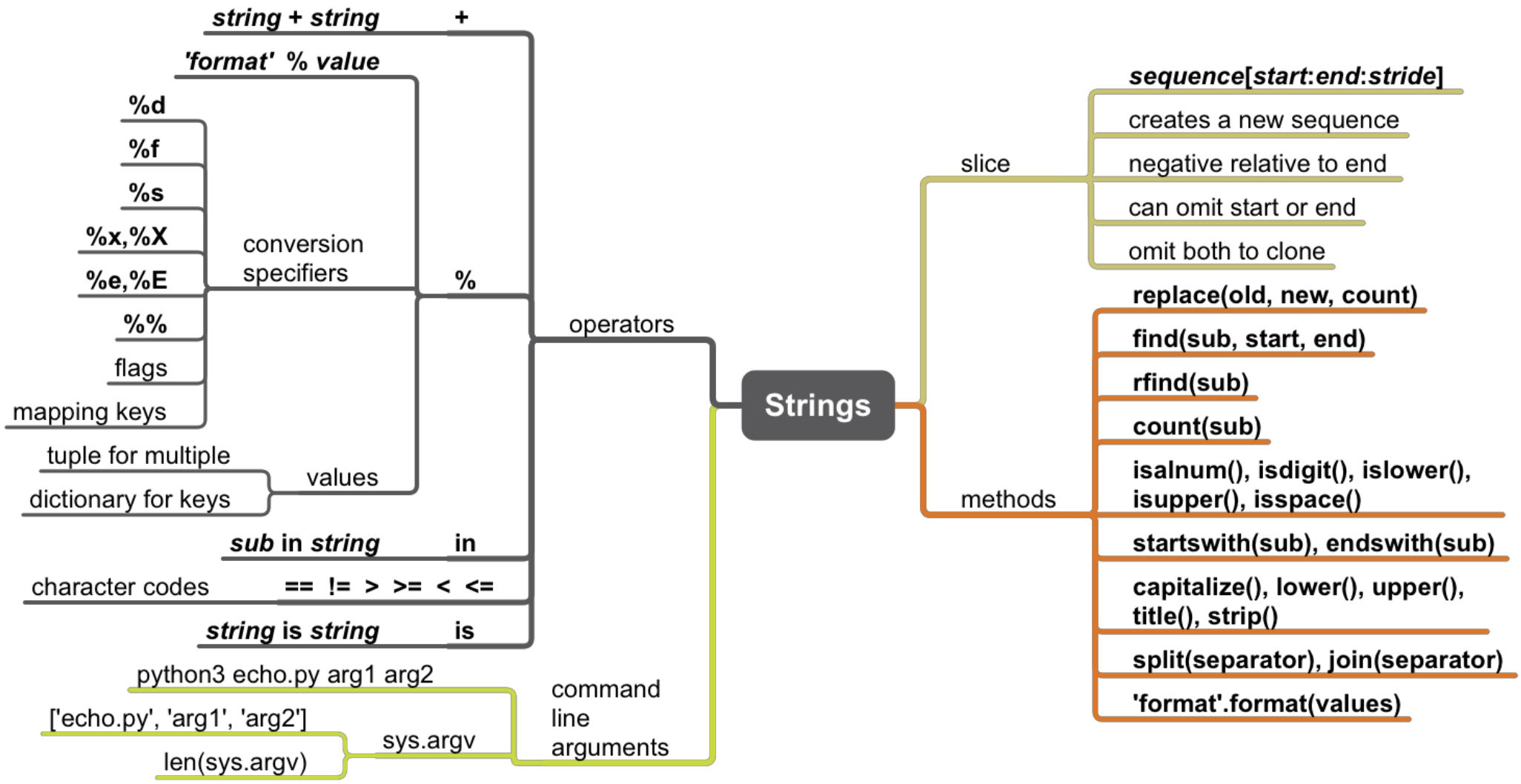

Functions
reasons
improve readability
encourage modular development
reduce redundancy
incremental development
polymorphism
behavior depends on argument types
dynamic typing
Python, Lisp
execution
static typing
Java, C, C++
compilation
scope
local
inside function
parameters, variables
locals()
global
in the file
variables
globals()
built-in
functions
standard library
math, random, time, datetime, ...
modules
scripts that may be imported
dot notation
__name__ == '__main__'
statements
def function(arguments):
indented block
global name
return value
import module
module.fn()
from module import fn
fn()
keyword arguments
map name instead of position
occur after positional arguments
def fn(a, b, c):
fn(1, 2, 3)
fn(1, c=2, b=3)
fn(c=1, b=2, a=3)
default parameter values
function calls may omit parameter
default used when omitted
def fn(a, b, c=0)
fn(1, 2)
fn(1, 2, 3)
multiple return values
return a container, unpack results
return (a, b, c)
x, y, z = fun()
objects
value is compiled bytecode
may be passed as parameter

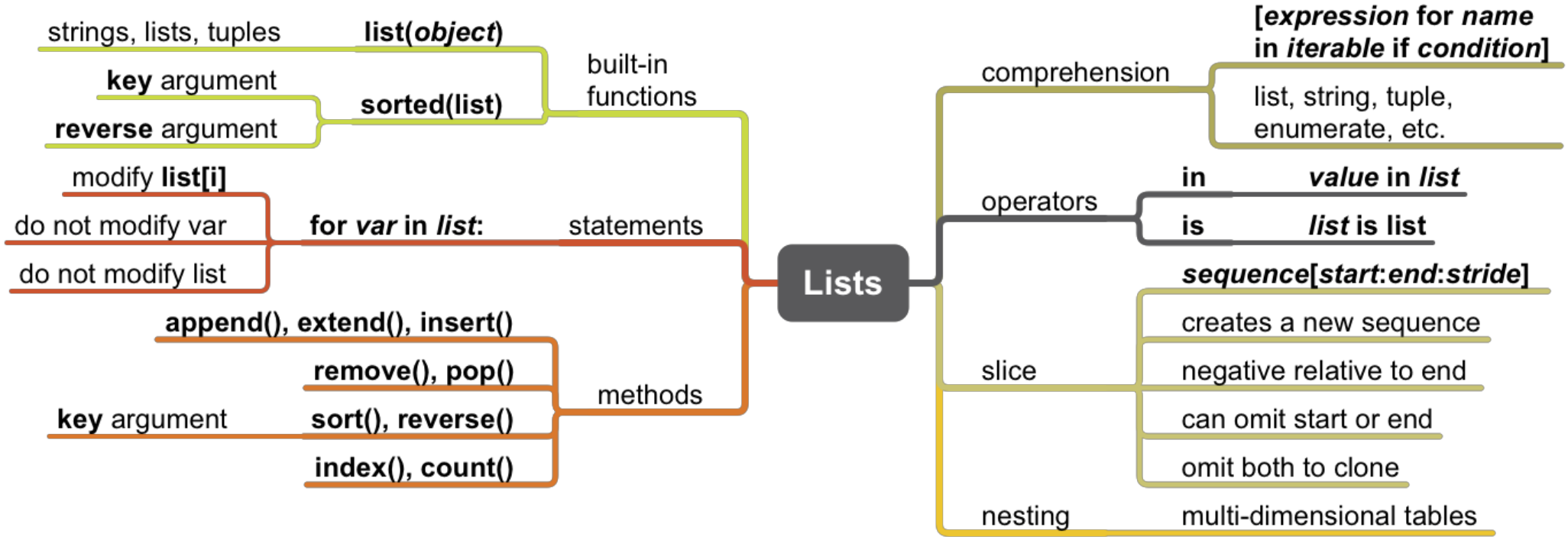

Decisions
syntax
statements
if condition:
elif condition:
else:
indentation
code blocks
nesting
expressions
value1 if condition else value2
condition precedence
values and functions
v, v[i], v[i:j:k], v.a, f(a)
arithmetic operators
() ** * / // % + -
comparisons (same)
==, !=, >, <, >=, <=
chained
in, not in
is, is not
booleans (ordered)
not
and
or

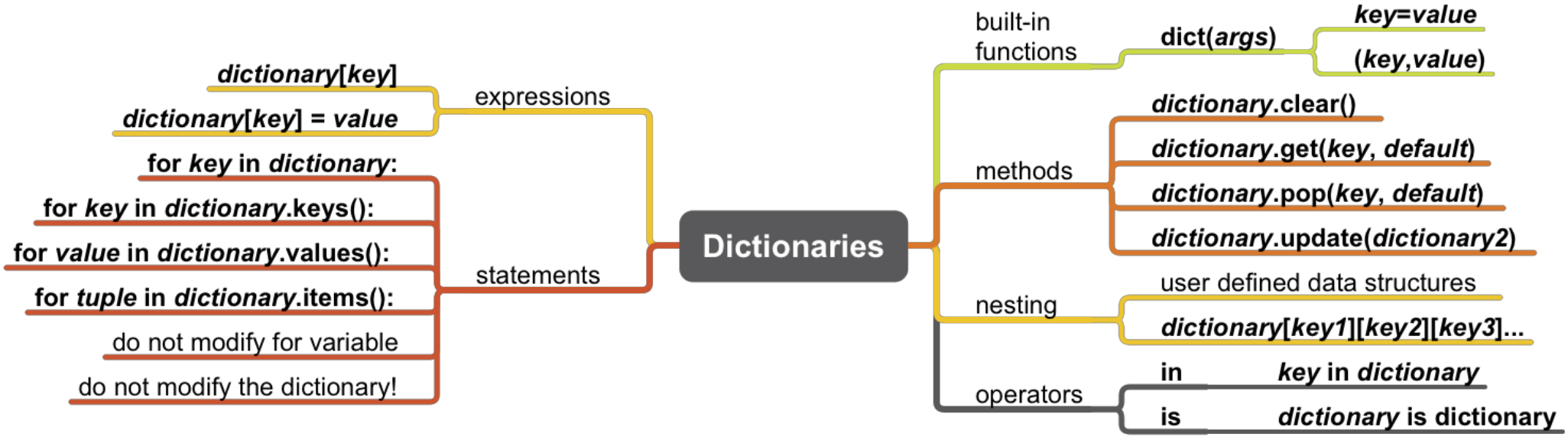

Repetition - while
loops
iterates same code over and over again
continuation / termination condition
statements
while *expression*:
unknown number of iterations
loop variable
count up
count down
update every iteration
step value
1<=N versus 0<N
infinite loop
criteria never met
indented statement block
nested loops

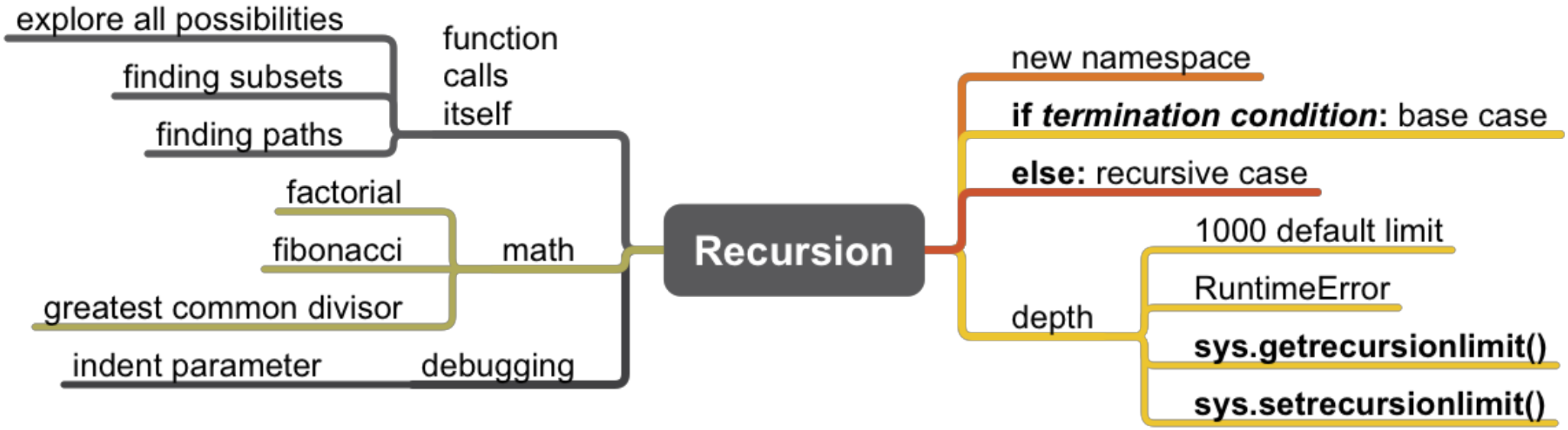

Repetition - for
guidelines
for
accessing elements of a container
string
characters
list, tuple
ordered values
dictionary
keys
number of iterations is computable
numbers
counting
range
guaranteed to complete
while
number of iterations is unknown
possible infinite loops
statements
for variable in container:
reversed(sequence)
range(limit)
range(start,limit,step)
enumerate(sequence)
unpacking
indented statement block
nested loops
break
immediate exit from loop
continue
immediate next loop iteration
else:
when normal loop termination

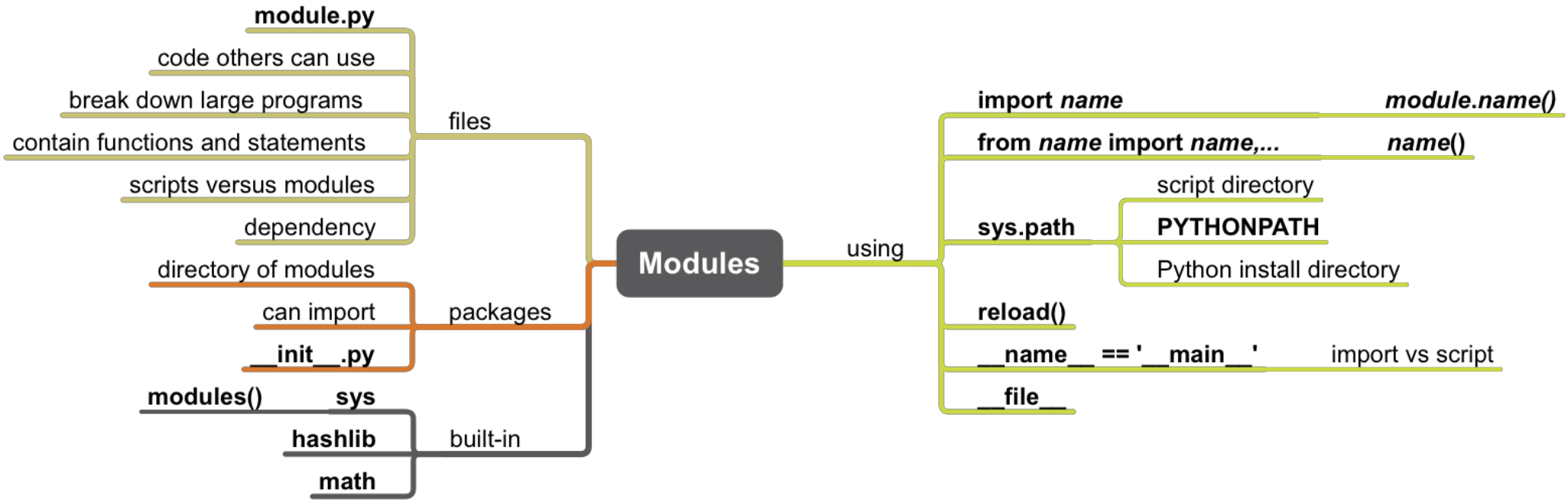

Files
os
module
operating system portability
win
\\
unix
/
mac
path
join()
split(path)
exists(path)
isfile(path)
getsize(path)
os.path
os.walk(path)
'b'
mode
big/little-endian
pack()
unpack()
struct
bytes()
binary
data
built-in functions
file = open(path,mode)
path
modes = r,w,a
file.close()
file.read()
file.readlines()
file.readline()
file.write(string)
file.flush()
statements
with open(path,mode) as file:
indented block
file automatically closed
csv
module
comma separated values files
list of strings
dictionary
list
rdr = csv.reader(file)
for rowlist in rdr:
wrtr = csv.writer(file)
wrtr.writerow(rowlist)
wrtr.writerows(rl1,rl2,...)
dict
rdr = csv.DictReader(...)
for rowdict in rdr:
wrtr = csv.DictWriter(...)
wrtr.writerow(rowdict)
wrtr.writerows(rd1,rd2,...)


Strings
operators
string + string
+
'format' % value
%
%d
%f
%s
%x,%X
%e,%E
%%
flags
mapping keys
conversion specifiers
tuple for multiple
dictionary for keys
values
sub in string
in
character codes
== != > >= < <=
string is string
is
command line arguments
python3 echo.py arg1 arg2
['echo.py', 'arg1', 'arg2']
len(sys.argv)
sys.argv
slice
sequence[start:end:stride]
creates a new sequence
negative relative to end
can omit start or end
omit both to clone
methods
replace(old, new, count)
find(sub, start, end)
rfind(sub)
count(sub)
isalnum(), isdigit(), islower(), isupper(), isspace()
startswith(sub), endswith(sub)
capitalize(), lower(), upper(), title(), strip()
split(separator), join(separator)
'format'.format(values)


Lists
built-in functions
strings, lists, tuples
list(object)
key argument
reverse argument
sorted(list)
statements
modify list[i]
do not modify var
do not modify list
for var in list:
methods
append(), extend(), insert()
remove(), pop()
key argument
sort(), reverse()
index(), count()
comprehension
[expression for name in iterable if condition]
list, string, tuple, enumerate, etc.
operators
in
value in list
is
list is list
slice
sequence[start:end:stride]
creates a new sequence
negative relative to end
can omit start or end
omit both to clone
nesting
multi-dimensional tables


Dictionaries
expressions
dictionary[key]
dictionary[key] = value
statements
for key in dictionary:
for key in dictionary.keys():
for value in dictionary.values():
for tuple in dictionary.items():
do not modify for variable
do not modify the dictionary!
built-in functions
dict(args)
key=value
(key,value)
methods
dictionary.clear()
dictionary.get(key, default)
dictionary.pop(key, default)
dictionary.update(dictionary2)
nesting
user defined data structures
dictionary[key1][key2][key3]...
operators
in
key in dictionary
is
dictionary is dictionary


Recursion
function calls itself
explore all possibilities
finding subsets
finding paths
math
factorial
fibonacci
greatest common divisor
debugging
indent parameter
new namespace
if termination condition: base case
else: recursive case
depth
1000 default limit
RuntimeError
sys.getrecursionlimit()
sys.setrecursionlimit()


Modules
files
module.py
code others can use
break down large programs
contain functions and statements
scripts versus modules
dependency
packages
directory of modules
can import
__init__.py
built-in
modules()
sys
hashlib
math
using
import name
module.name()
from name import name,...
name()
sys.path
script directory
PYTHONPATH
Python install directory
reload()
__name__ == '__main__'
import vs script
__file__