W20 CS 116

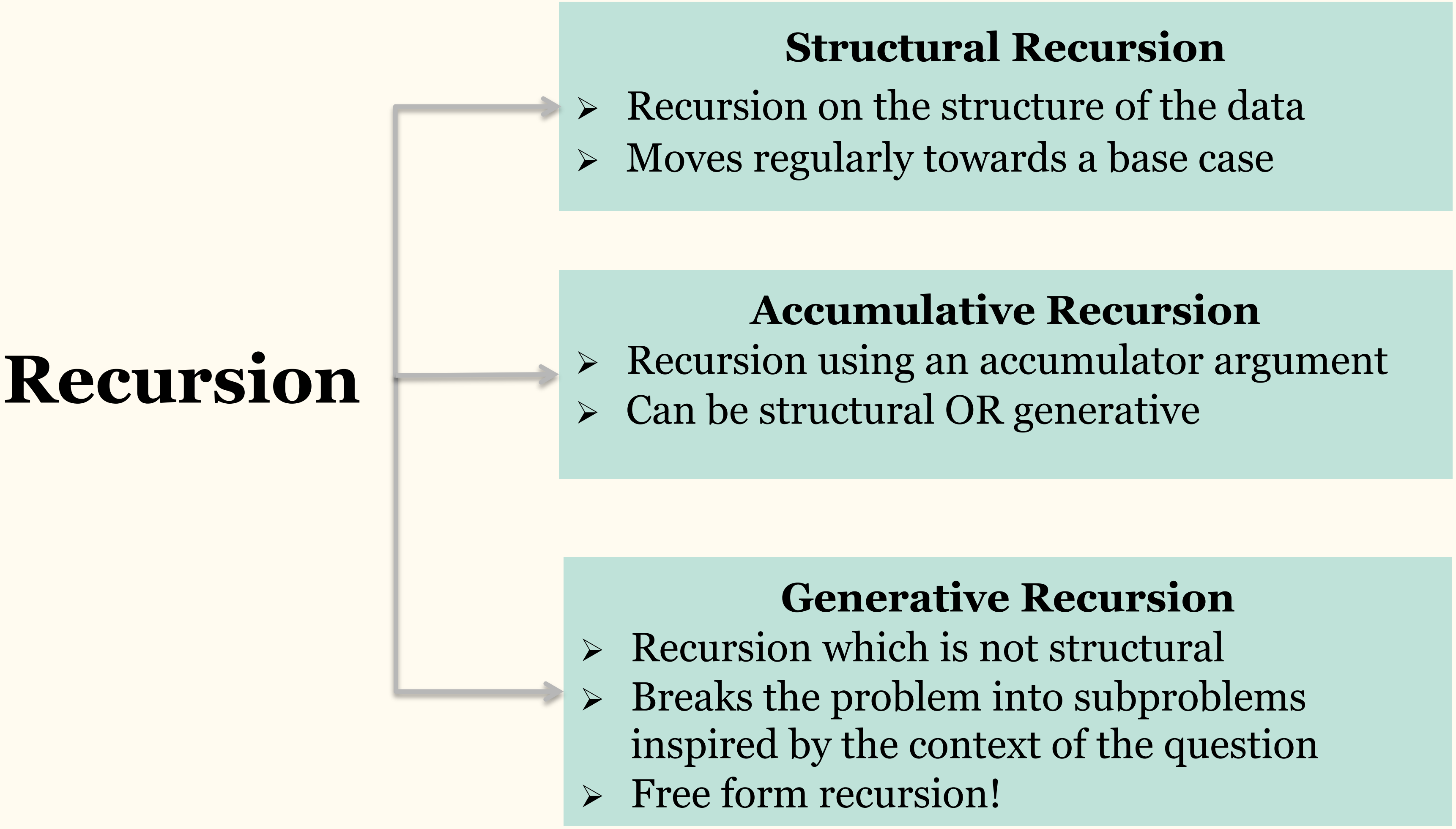
Midterm Q&A Session

Midterm Information

The CS116 Midterm

- Monday March 2nd, 2020, 7:00–8:50 p.m.
- Look up your writing location on Odyssey
- Covers Module 1-5 and Assignment 1-5 (No loops)

Recursion



Recursion

Structural Recursion

- Recursion on the structure of the data
- Moves regularly towards a base case

Accumulative Recursion

- Recursion using an accumulator argument
- Can be structural OR generative

Generative Recursion

- Recursion which is not structural
- Breaks the problem into subproblems inspired by the context of the question
- Free form recursion!

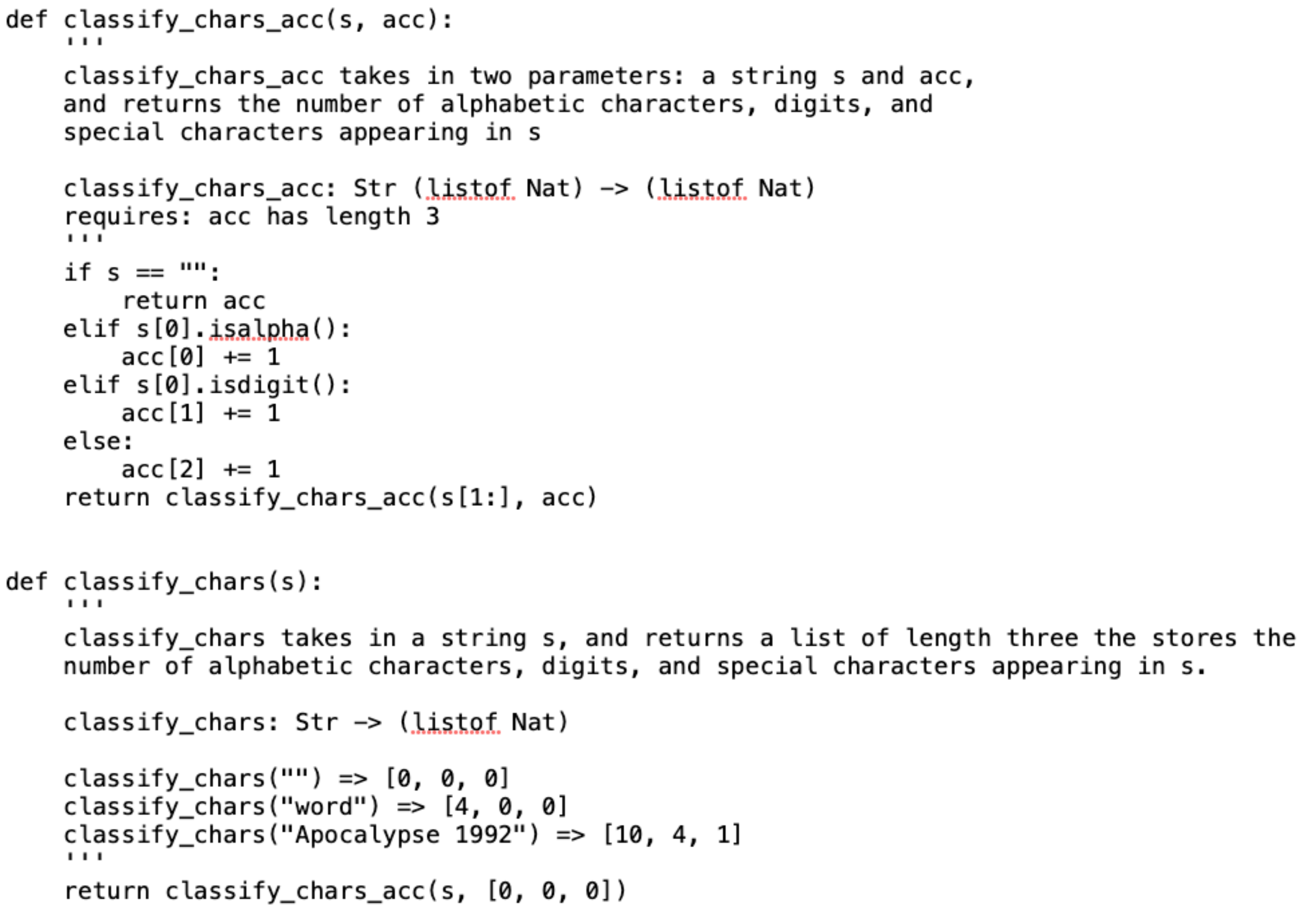
Accumulative Example

Use accumulative recursion to complete the function `classify_chars`, that consumes a string `s`, and produces a list containing exactly three natural numbers:

- the number of alphabetic characters in `s`
- the number of digits in `s`
- the number of all other characters in `s`

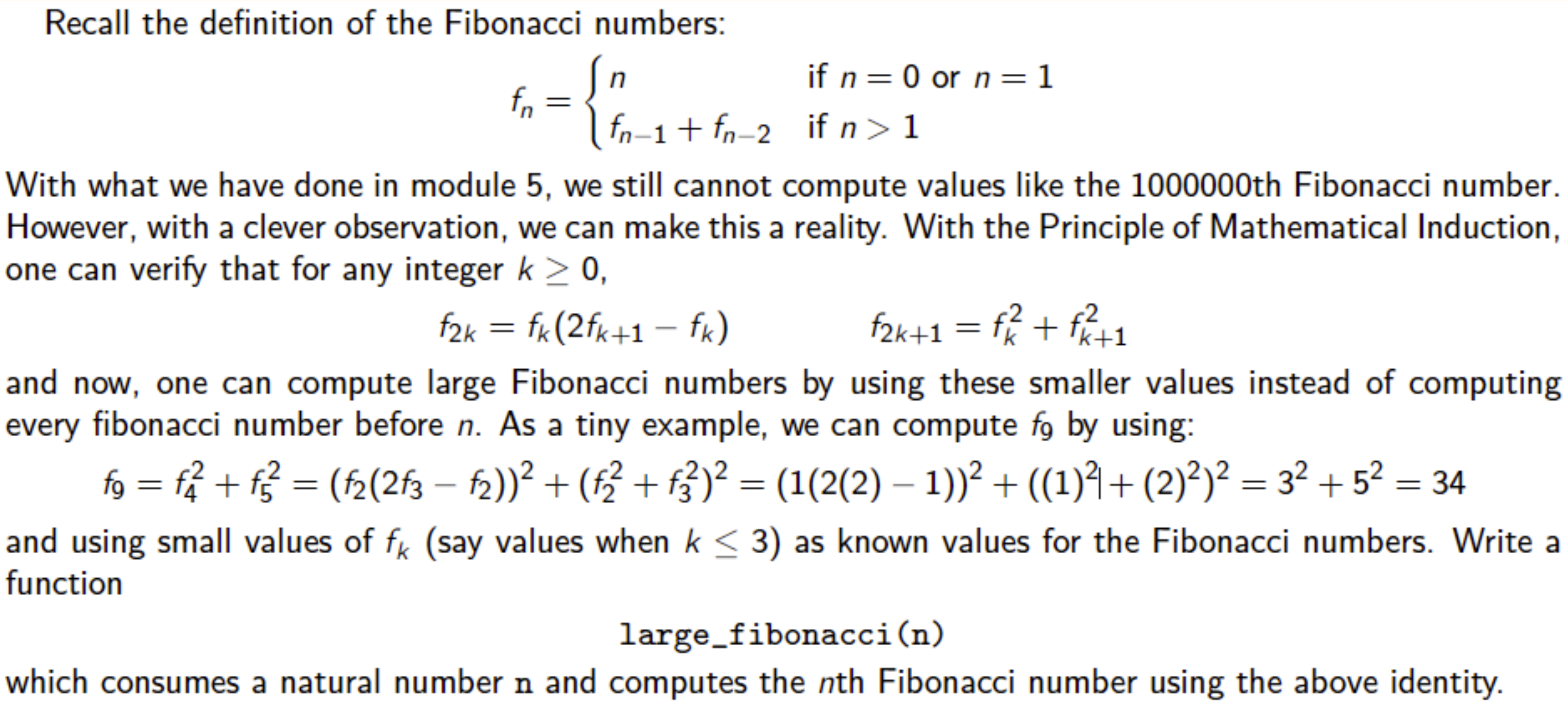
For example,

- `classify_chars("Hello CS116") => [7, 3, 1]`
- `classify_chars("2*math.pi*r") => [7, 1, 3]`



```
def classify_chars_acc(s, acc):
    '''
    classify_chars_acc takes in two parameters: a string s and acc,
    and returns the number of alphabetic characters, digits, and
    special characters appearing in s

    classify_chars_acc: Str (listof Nat) -> (listof Nat)
    requires: acc has length 3
    '''
    if s == "":
        return acc
    elif s[0].isalpha():
        acc[0] += 1
    elif s[0].isdigit():
        acc[1] += 1
    else:
        acc[2] += 1
    return classify_chars_acc(s[1:], acc)


def classify_chars(s):
    '''
    classify_chars takes in a string s, and returns a list of length three the stores the
    number of alphabetic characters, digits, and special characters appearing in s.

    classify_chars: Str -> (listof Nat)

    classify_chars("") => [0, 0, 0]
    classify_chars("word") => [4, 0, 0]
    classify_chars("Apocalypse 1992") => [10, 4, 1]
    '''
    return classify_chars_acc(s, [0, 0, 0])
```

Generative Example



Recall the definition of the Fibonacci numbers:

$$f_n = \begin{cases} n & \text{if } n = 0 \text{ or } n = 1 \\ f_{n-1} + f_{n-2} & \text{if } n > 1 \end{cases}$$

With what we have done in module 5, we still cannot compute values like the 1000000th Fibonacci number. However, with a clever observation, we can make this a reality. With the Principle of Mathematical Induction, one can verify that for any integer $k \geq 0$,

$$f_{2k} = f_k(2f_{k+1} - f_k) \qquad\qquad f_{2k+1} = f_k^2 + f_{k+1}^2$$

and now, one can compute large Fibonacci numbers by using these smaller values instead of computing every fibonacci number before n. As a tiny example, we can compute f_9 by using:

$$f_9 = f_4^2 + f_5^2 = (f_2(2f_3 - f_2))^2 + (f_2^2 + f_3^2)^2 = (1(2(2) - 1))^2 + ((1)^2 + (2)^2)^2 = 3^2 + 5^2 = 34$$

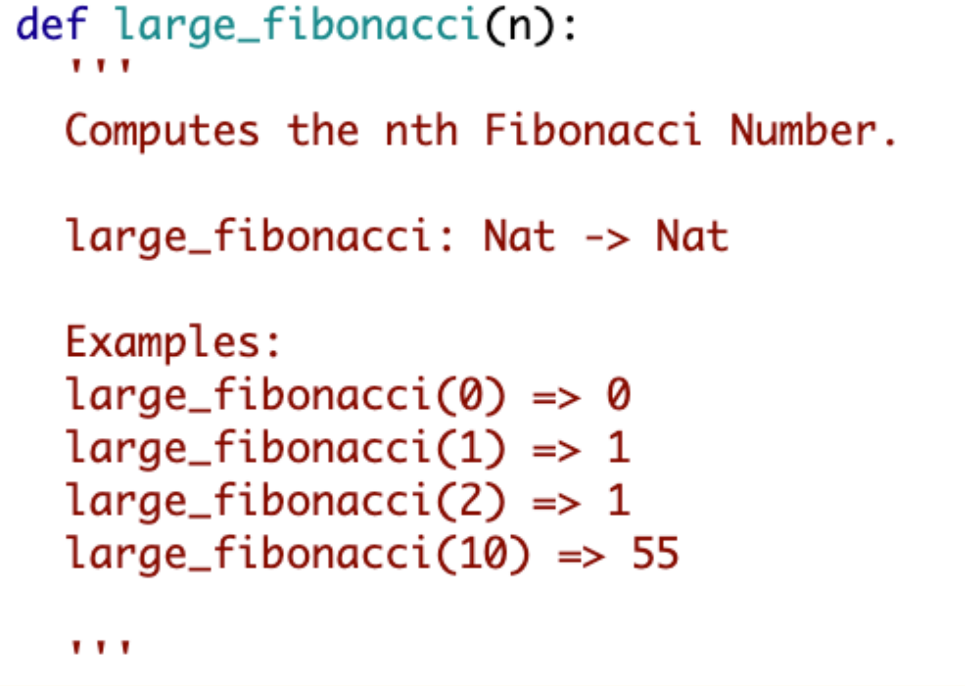
and using small values of f_k (say values when $k \leq 3$) as known values for the Fibonacci numbers. Write a function

```
large_fibonacci(n)
```

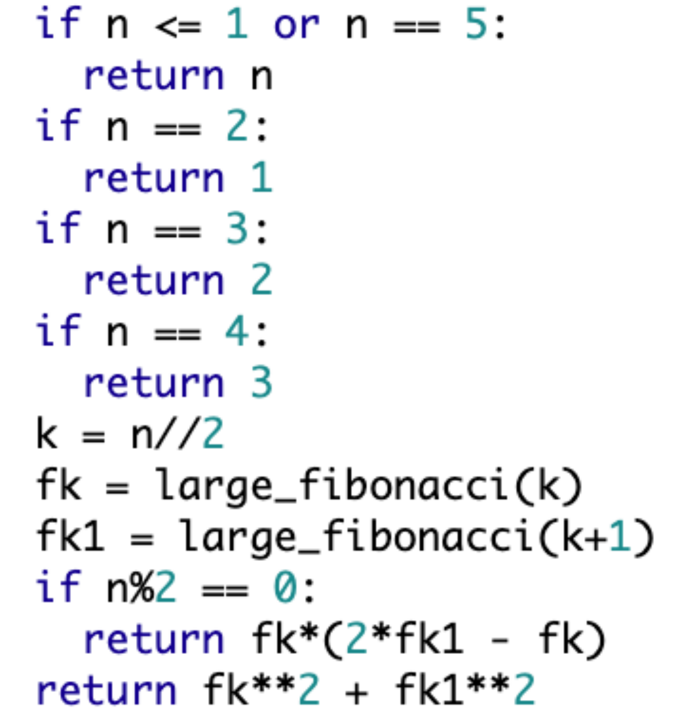
which consumes a natural number `n` and computes the nth Fibonacci number using the above identity.

Hint: You must store the values of fk and fk+1 in variables before attempting to compute either f2k or f2k+1. Recomputing these values will cause your code to time out.

Solution



```
def large_fibonacci(n):
  '''
  Computes the nth Fibonacci Number.

  large_fibonacci: Nat -> Nat

  Examples:
  large_fibonacci(0) => 0
  large_fibonacci(1) => 1
  large_fibonacci(2) => 1
  large_fibonacci(10) => 55

  '''
```



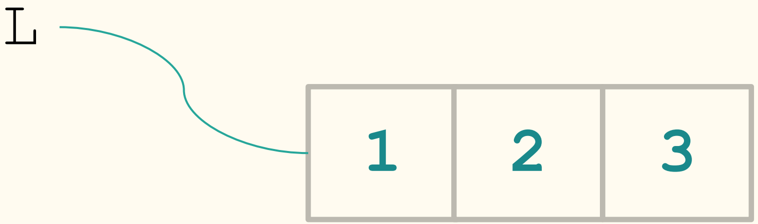
```
if n <= 1 or n == 5:
  return n
if n == 2:
  return 1
if n == 3:
  return 2
if n == 4:
  return 3
k = n//2
fk = large_fibonacci(k)
fk1 = large_fibonacci(k+1)
if n%2 == 0:
  return fk*(2*fk1 - fk)
return fk**2 + fk1**2
```

Mutation

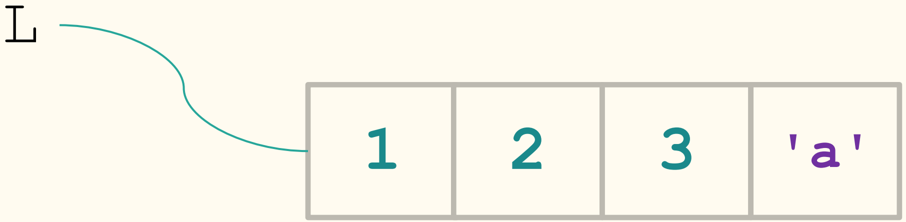
Tracing

```
L = [1,2,3]
L.append('a')             # What is value of L?
M = L                     # M & L are aliases. Why?
M.extend([True, False])   # What is the value of L? M?
L.insert(3,'new')         # What is the value of L? M?
L = []                    # What is the value of L? M? Are they aliases?
M.append(1)               # What is the value of M?
M.remove(1)               # What is the value of M?
x = M.pop(1)
```

```
L = [1,2,3]
```



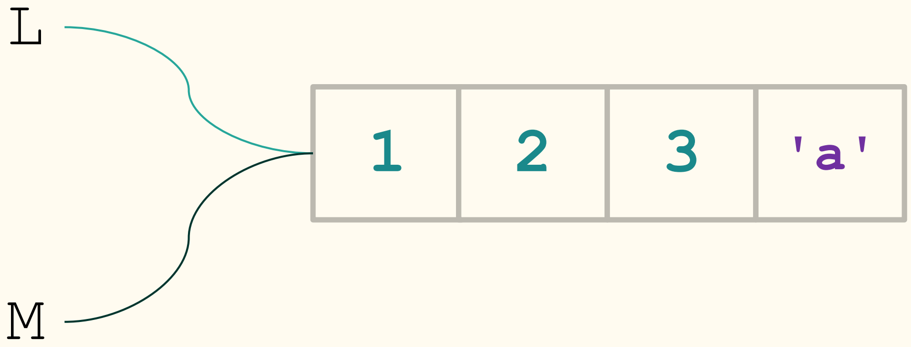
```
L = [1,2,3]
L.append('a')
```



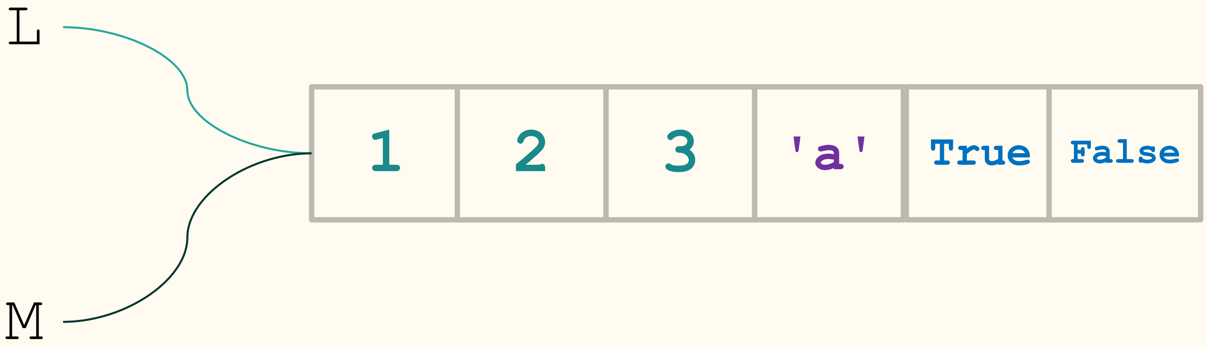
L

1 2 3 'a'

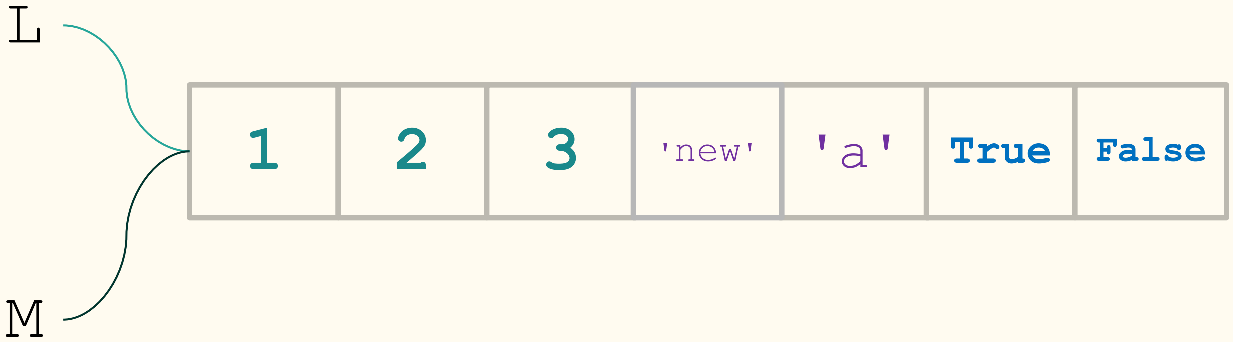
```
L = [1,2,3]
L.append('a')
M = L
```



```
L = [1,2,3]
L.append('a')
M = L
M.extend([True, False])
```



```
L.insert(3,'new')
```

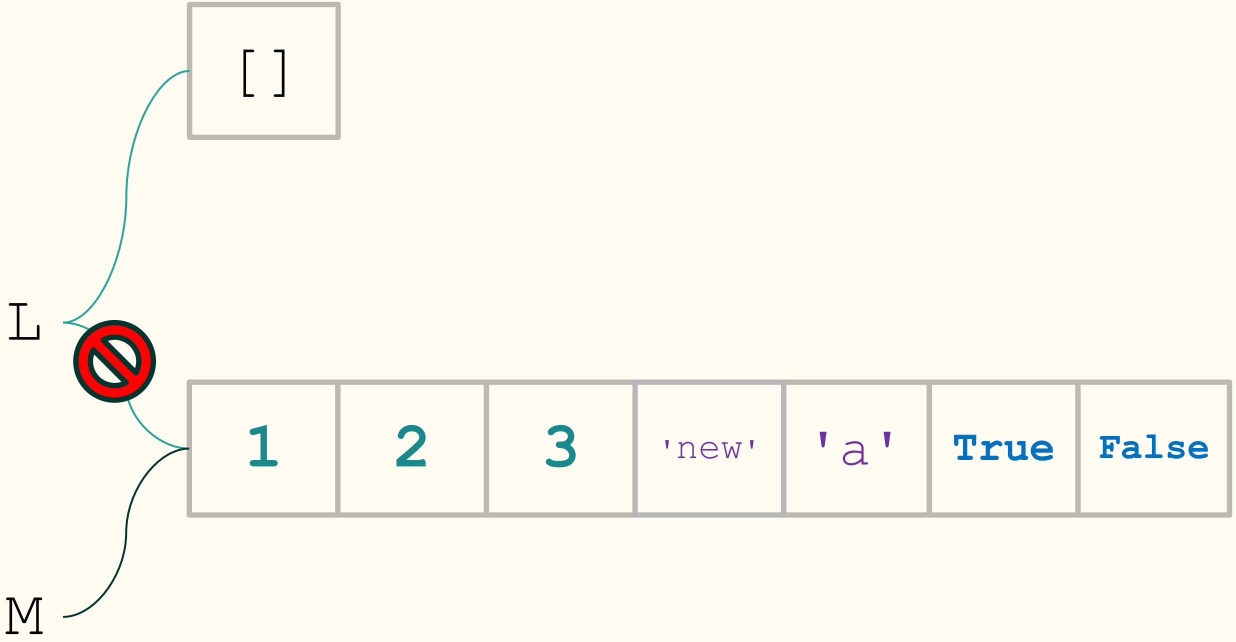


L

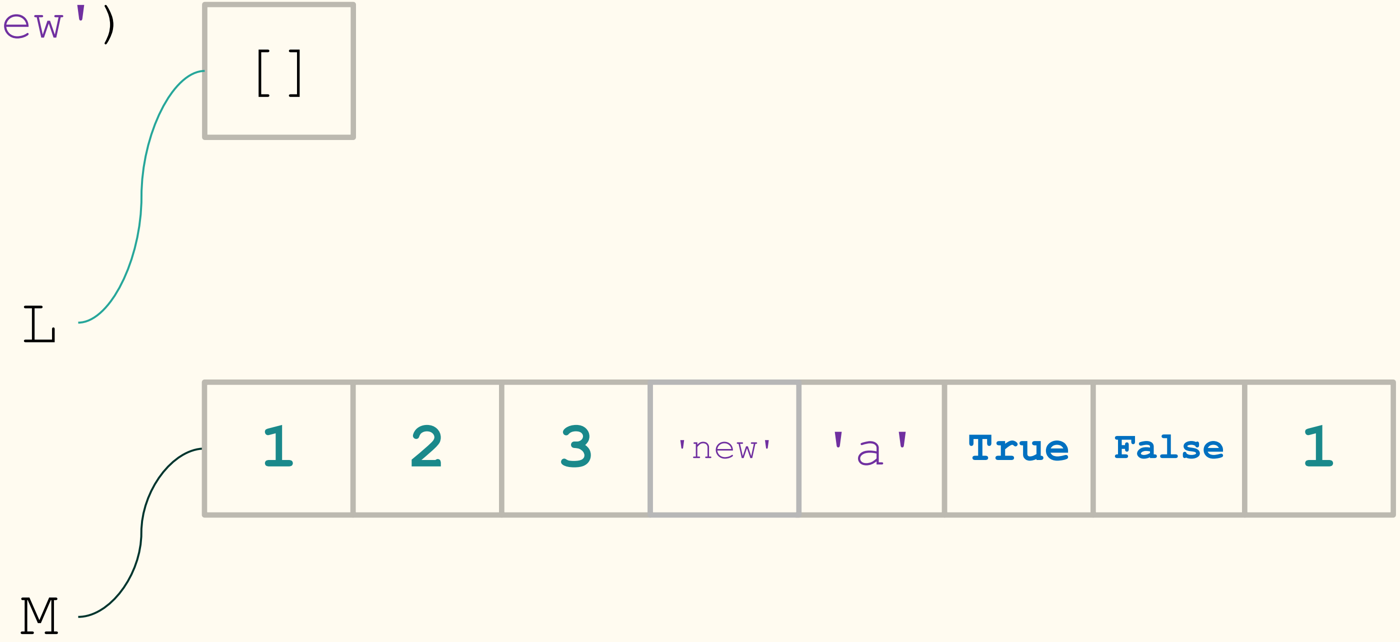
M

1	2	3	'new'	'a'	True	False

```
L.insert(3,'new')
L = []
```



```
L.insert(3,'new')
L = []
M.append(1)
```



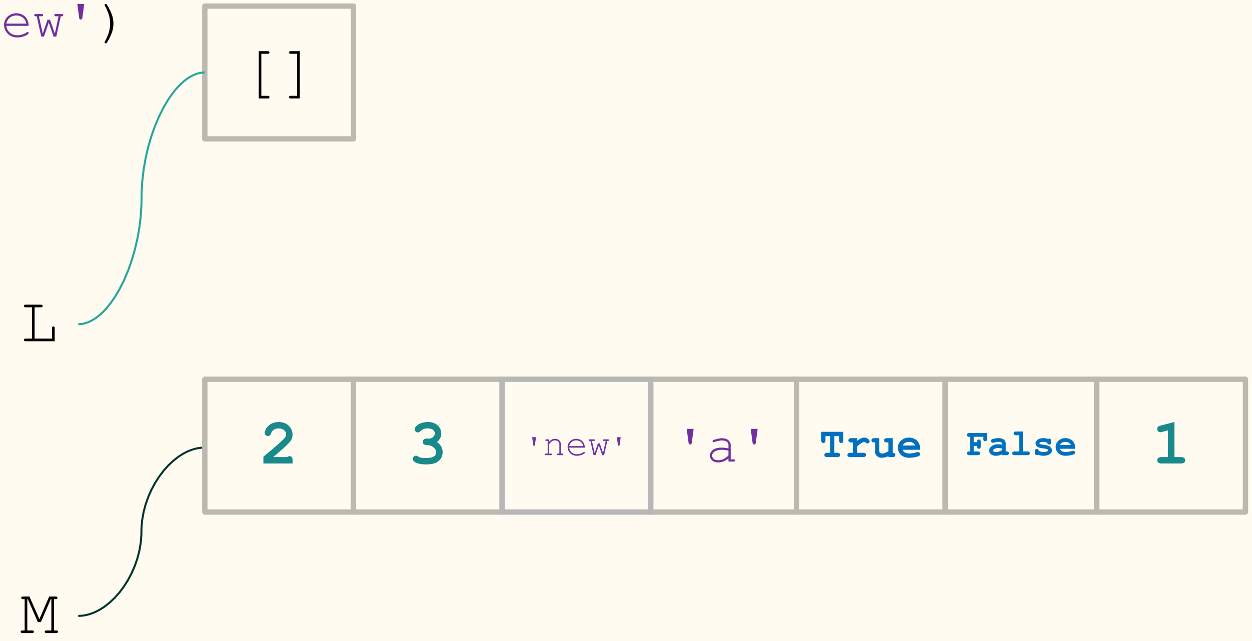
[]

L

1	2	3	'new'	'a'	True	False	1

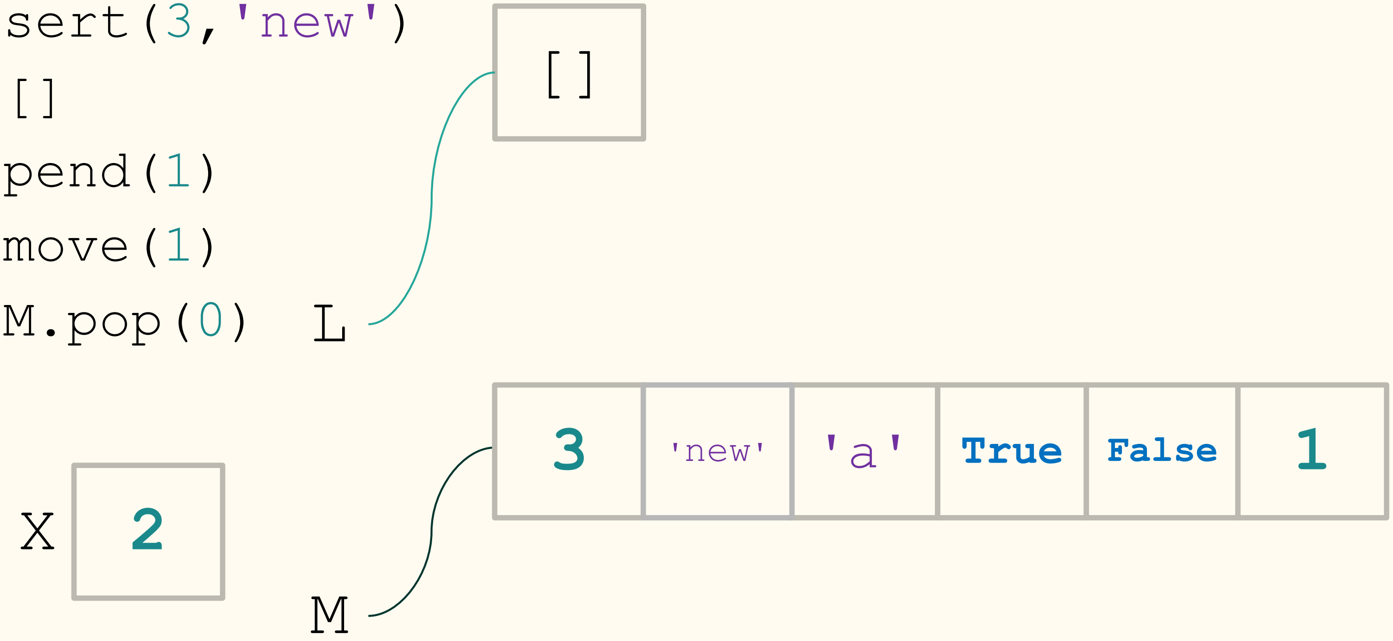
M

```
L.insert(3,'new')
L = []
M.append(1)
M.remove(1)
```



2	3	'new'	'a'	True	False	1

```
L.insert(3,'new')
L = []
M.append(1)
M.remove(1)
X = M.pop(0)
```



sorted_insert()

For many applications, it is worth the effort to keep a list in sorted order. Write a function `sorted_insert` that takes in an alphabetically-ordered list of strings, `los`, and a (possibly disordered) list of string `new_los`, and mutates los by inserting each element of `new_los` into its appropriate place in `los`.

For example:

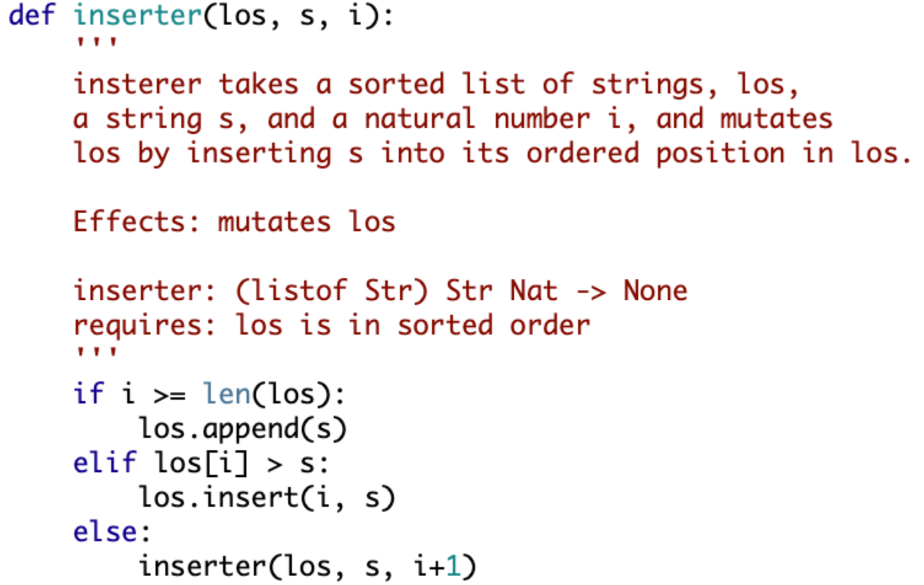
```
L = [“stuff”, “words”]
sorted_insert(L, []) => None
```

and `L` becomes `[“stuff”, “words”]`

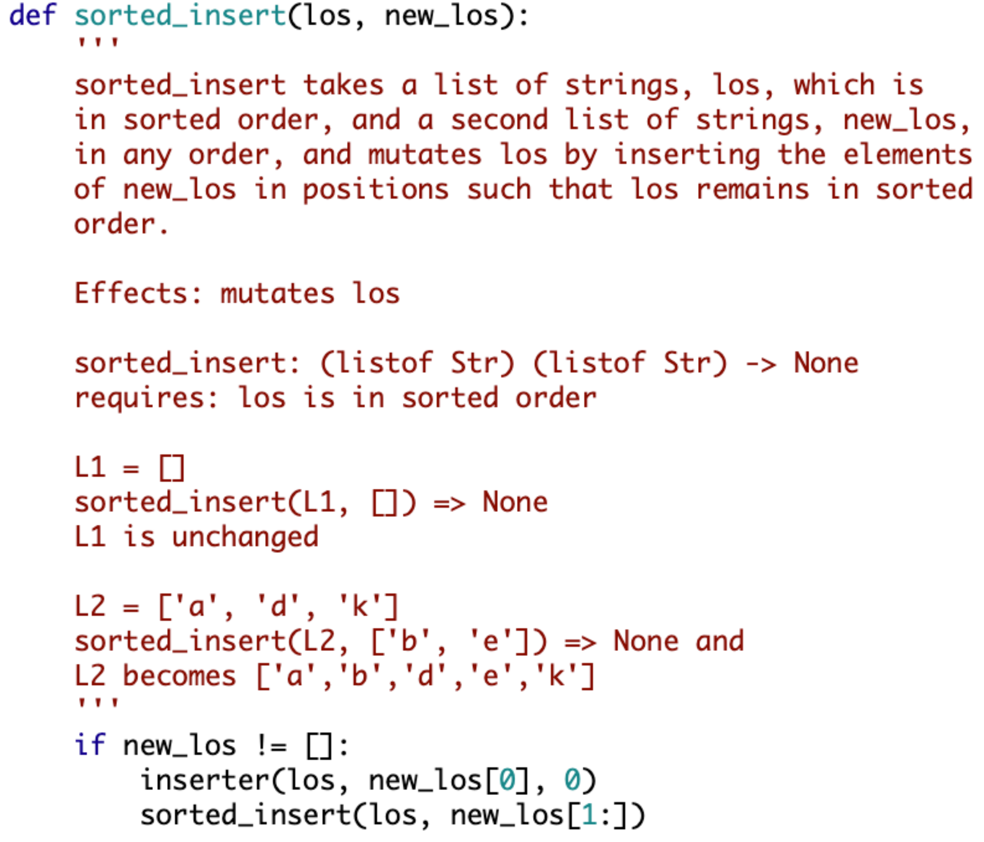
sorted_insert() - Examples

```
# Ex1
L = [“hello”, “world”]
sorted_insert(L, [“computer”, “artificial”]) => None
and L becomes [“artificial”, “computer”, “hello”, “world”]
# Ex2
L = [“Angus”, “Hoots”, “Zargothrax”]
sorted_insert(L, [“Ralathor”, “Christopher”]) => None
and L becomes [“Angus”, “Christopher”, “Hoots”, “Ralathor”, “Zargothrax”]
# Ex3
M = []
sorted_insert(M, [“stringy”]) => None
and M becomes [“stringy”]
```

Solution



```
def inserter(los, s, i):
    '''
    insterer takes a sorted list of strings, los,
    a string s, and a natural number i, and mutates
    los by inserting s into its ordered position in los.

    Effects: mutates los

    inserter: (listof Str) Str Nat -> None
    requires: los is in sorted order
    '''
    if i >= len(los):
        los.append(s)
    elif los[i] > s:
        los.insert(i, s)
    else:
        inserter(los, s, i+1)
```



```
def sorted_insert(los, new_los):
    '''
    sorted_insert takes a list of strings, los, which is
    in sorted order, and a second list of strings, new_los,
    in any order, and mutates los by inserting the elements
    of new_los in positions such that los remains in sorted
    order.

    Effects: mutates los

    sorted_insert: (listof Str) (listof Str) -> None
    requires: los is in sorted order

    L1 = []
    sorted_insert(L1, []) => None
    L1 is unchanged

    L2 = ['a', 'd', 'k']
    sorted_insert(L2, ['b', 'e']) => None and
    L2 becomes ['a','b','d','e','k']
    '''
    if new_los != []:
        inserter(los, new_los[0], 0)
        sorted_insert(los, new_los[1:])
```

Design Recipe

Purpose

- Brief description of what the function does.

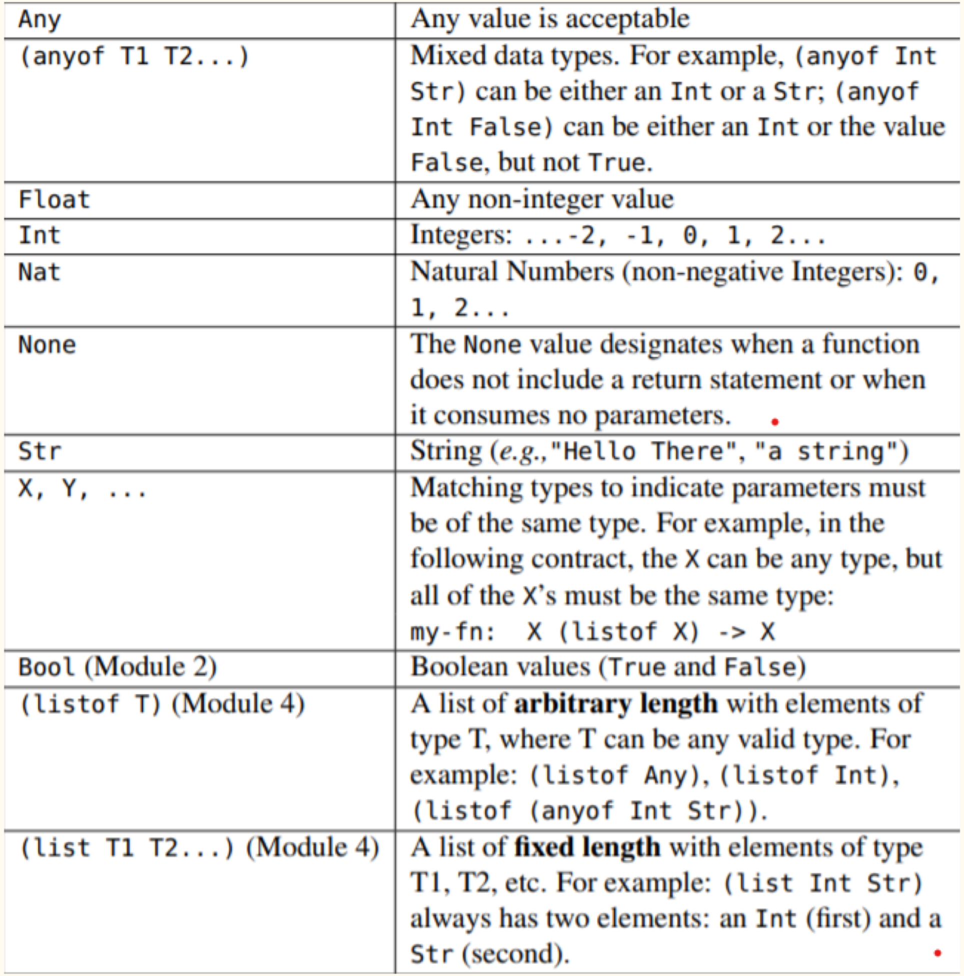
- Important!

  Include the parameter names to show the relationship between input and the function’s actions

Contract and Requirements

- Refer to next slide (see style guide too!) for types allowed
- Use single arrows
- No Num type in CS 116!
  - If your function consumes integers and decimal numbers, use (anyof Int Float)
- Be as specific as possible.
  - If your function consumes a non-negative integer n:

    Int v.s. Nat



`Any`	Any value is acceptable
`(anyof T1 T2...)`	Mixed data types. For example, `(anyof Int Str)` can be either an `Int` or a `Str`; `(anyof Int False)` can be either an `Int` or the value `False`, but not `True`.
`Float`	Any non-integer value
`Int`	Integers: `...-2, -1, 0, 1, 2...`
`Nat`	Natural Numbers (non-negative Integers): `0, 1, 2...`
`None`	The `None` value designates when a function does not include a return statement or when it consumes no parameters.
`Str`	String (*e.g.*,`"Hello There"`, `"a string"`)
`X, Y, ...`	Matching types to indicate parameters must be of the same type. For example, in the following contract, the `X` can be any type, but all of the `X`'s must be the same type: `my-fn:  X (listof X) -> X`
`Bool` (Module 2)	Boolean values (`True` and `False`)
`(listof T)` (Module 4)	A list of **arbitrary length** with elements of type T, where T can be any valid type. For example: `(listof Any)`, `(listof Int)`, `(listof (anyof Int Str))`.
`(list T1 T2...)` (Module 4)	A list of **fixed length** with elements of type T1, T2, etc. For example: `(list Int Str)` always has two elements: an `Int` (first) and a `Str` (second).

Effects

- Used whenever the function changes the Python environment, i.e, does something else other than returning a value.
- Effects section is included **after** the purpose and **before** the contract.
- Include effects for helper functions too!

- Types of Effects:

  1) Reading in user input from keyboard
  2) Printing to the screen
  3) Mutating a list

Examples

- Include at least 2 examples
- Include both some typical cases and edge cases:
  - Edge case examples:
    - Empty list if your function can consume it.
    - 0 if this is the smallest number your function can consume.
    - Empty string if your function can consume it.
    - True and False cases if your function returns Boolean values

Example 1

Write a function, fn_1 that consumes a list of integers, L and mutates L such that all the even elements in L are divided by 2, and the odd elements in L are doubled.

```
 ''' Mutates a list of integers, L such that even numbers are
halved and odd numbers are doubled.

Effect: Mutates L

fn_1: (listof Int) -> None

Examples:

L=[] ; fn_1(L) => None ; L is mutated to []

M=[0]; fn_1(L) => None; M is mutated to [0]

N = [-2,5] ; fn_1 (N) => None; N is mutated to [-1,10] '''
```

Example 2

Write a function, fn_2, that consumes a list of integers, L and a natural number, n and returns a list which contains all the elements of L multiplied by n.

```
''' Returns a list which consists of all the elements of L, a
list of integers, multiplied by n, a natural number.

fn_2: (listof Int) Nat -> (listof Int)

Examples:

fn_2([],3) => []

fn_2([0],4) => [0]

fn_2([-2,4],3) => [-6,12] '''
```

Testing Code

Functions

```
# test for returning result except Float
check.expect(“label1”, fn_name1(x1, x2, …),expected_result)

# test for returning result of Float
check.within(“label2”, fn_name2(x1, x2, …),expected_result,tolerane)

# when the function asks for user input
check.set_input(“input 1”, “input 2”, …)

# when the function prints. Use one of:
check.set_print_exact(“Printed at line1.\nPrinted at line2.\n…”)
check.set_screen(“description of the printing result”)
```

Testing for Mutation

```
L = […, …, …, ……]
check.expect(“test label-return”, fn_name(L), output)
check.expect(“test label-mutation”, L, [mutation result])
```

Example: test for sorted_insert

```
## L = [“words”, “stuff”]
## sorted_insert(L, []) => None
## and L becomes [“words”, “stuff”]

# Test 1
L = [“stuff”, “words”]
check.expect(“test 1 - return”, sorted_insert(L,[]), None)
check.expect(“test 1 - mutation”, L, [“stuff”, “words”])
```

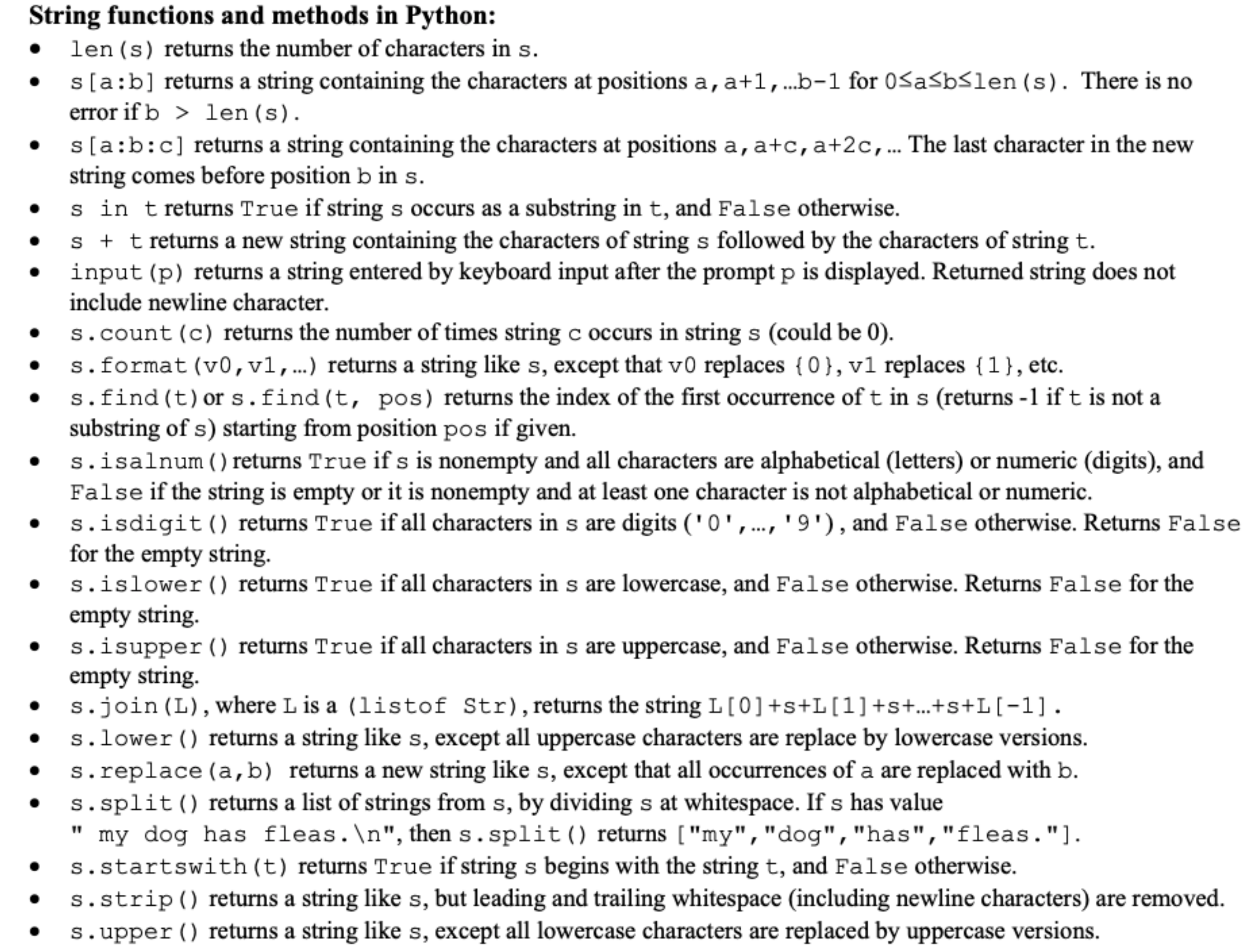
Testing for input() and print()

```
check.set_input(“input 1”, “input 2”, …)
check.set_print_exact(“Printed at line1.\nPrinted at line2.\n…”)
check.expect(”test label”, fn_name(), expected_result)
```

Example: test for mastermind

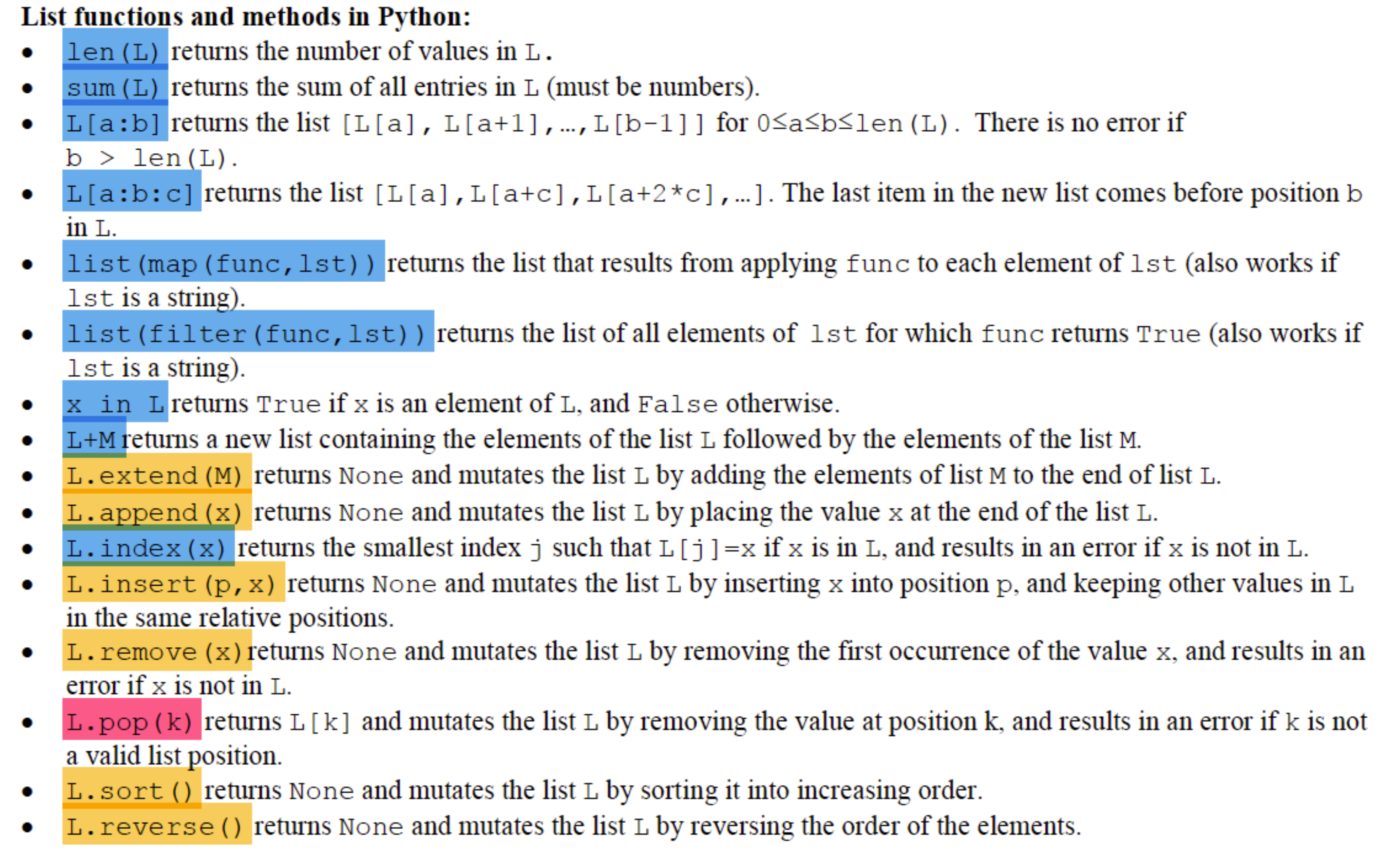
```
# mastermind
check.set_input(“1 2 3 4”, “6 2 3 5”, “1 6 1 4”)

check.set_print_exact(“There are 2 numbers in correct places and 0 numbers in incorrect places .\nThere are 0 numbers in correct places and 1 numbers in incorrect places .\nSequence found in 3 guesses .”)

check.expect(“mastermind_1”, mastermind([1, 6, 1, 4], 5), 3)
```

String and list methods



String functions and methods in Python:

- `len(s)` returns the number of characters in `s`.
- `s[a:b]` returns a string containing the characters at positions `a,a+1,…b-1` for `0≤a≤b≤len(s)`. There is no error if `b > len(s)`.
- `s[a:b:c]` returns a string containing the characters at positions `a,a+c,a+2c,…` The last character in the new string comes before position `b` in `s`.
- `s in t` returns `True` if string `s` occurs as a substring in `t`, and `False` otherwise.
- `s + t` returns a new string containing the characters of string `s` followed by the characters of string `t`.
- `input(p)` returns a string entered by keyboard input after the prompt `p` is displayed. Returned string does not include newline character.
- `s.count(c)` returns the number of times string `c` occurs in string `s` (could be 0).
- `s.format(v0,v1,…)` returns a string like `s`, except that `v0` replaces `{0}`, `v1` replaces `{1}`, etc.
- `s.find(t)` or `s.find(t, pos)` returns the index of the first occurrence of `t` in `s` (returns -1 if `t` is not a substring of `s`) starting from position `pos` if given.
- `s.isalnum()` returns `True` if `s` is nonempty and all characters are alphabetical (letters) or numeric (digits), and `False` if the string is empty or it is nonempty and at least one character is not alphabetical or numeric.
- `s.isdigit()` returns `True` if all characters in `s` are digits `('0',…,'9')`, and `False` otherwise. Returns `False` for the empty string.
- `s.islower()` returns `True` if all characters in `s` are lowercase, and `False` otherwise. Returns `False` for the empty string.
- `s.isupper()` returns `True` if all characters in `s` are uppercase, and `False` otherwise. Returns `False` for the empty string.
- `s.join(L)`, where `L` is a `(listof Str)`, returns the string `L[0]+s+L[1]+s+…+s+L[-1]`.
- `s.lower()` returns a string like `s`, except all uppercase characters are replace by lowercase versions.
- `s.replace(a,b)` returns a new string like `s`, except that all occurrences of `a` are replaced with `b`.
- `s.split()` returns a list of strings from `s`, by dividing `s` at whitespace. If `s` has value `" my dog has fleas.\n"`, then `s.split()` returns `["my","dog","has","fleas."]`.
- `s.startswith(t)` returns `True` if string `s` begins with the string `t`, and `False` otherwise.
- `s.strip()` returns a string like `s`, but leading and trailing whitespace (including newline characters) are removed.
- `s.upper()` returns a string like `s`, except all lowercase characters are replaced by uppercase versions.



List functions and methods in Python:

- `len(L)` returns the number of values in `L`.
- `sum(L)` returns the sum of all entries in `L` (must be numbers).
- `L[a:b]` returns the list `[L[a], L[a+1],…,L[b-1]]` for `0≤a≤b≤len(L)`. There is no error if `b > len(L)`.
- `L[a:b:c]` returns the list `[L[a],L[a+c],L[a+2*c],…]`. The last item in the new list comes before position `b` in `L`.
- `list(map(func,lst))` returns the list that results from applying `func` to each element of `lst` (also works if `lst` is a string).
- `list(filter(func,lst))` returns the list of all elements of `lst` for which `func` returns `True` (also works if `lst` is a string).
- `x in L` returns `True` if `x` is an element of `L`, and `False` otherwise.
- `L+M` returns a new list containing the elements of the list `L` followed by the elements of the list `M`.
- `L.extend(M)` returns `None` and mutates the list `L` by adding the elements of list `M` to the end of list `L`.
- `L.append(x)` returns `None` and mutates the list `L` by placing the value `x` at the end of the list `L`.
- `L.index(x)` returns the smallest index `j` such that `L[j]=x` if `x` is in `L`, and results in an error if `x` is not in `L`.
- `L.insert(p,x)` returns `None` and mutates the list `L` by inserting `x` into position `p`, and keeping other values in `L` in the same relative positions.
- `L.remove(x)` returns `None` and mutates the list `L` by removing the first occurrence of the value `x`, and results in an error if `x` is not in `L`.
- `L.pop(k)` returns `L[k]` and mutates the list `L` by removing the value at position k, and results in an error if `k` is not a valid list position.
- `L.sort()` returns `None` and mutates the list `L` by sorting it into increasing order.
- `L.reverse()` returns `None` and mutates the list `L` by reversing the order of the elements.