CSC148-Section:L0301
Week#6-Wednesday

Instructed by
AbdulAziz Al-Helali
a.alhelali@mail.utoronto.ca
Office hours: Wednesday 11-1, BA2230.

Slides adapted from Professor Danny Heap course material
winter17
Outline

• Recursion
  • Quick review of recursion and the template for writing recursive functions
  • Implement recursive functions
Template for structural recursion

recursion when input is a recursive structure:

• Base case
  • if input cannot be decomposed into recursive sub-structures, you have a base case and you directly return a result without recursion

• General case
  • if input can be decomposed into recursive sub-structures, solve them recursively and combine the result(s)
Template

```
if isinstance(list_, list): # General case
    return helper_function([your_rec_func(x) for x in list_])
else: # Base case
    return list_
```

Use List comprehensions

Do something with base input
• **expression 1** is evaluated if condition is True
• **expression 2** is evaluated if condition is False
```python
def gather_lists(list_: Union[List, object]) -> List[object]:
    """
    Return the concatenation of the sublists of list_.
    
    >>> list_ = [[1, 2], [3, 4]]
    >>> gather_lists(list_)
    [1, 2, 3, 4]
    >>> list_ = [[1,[ -1,0],2], [3, 4]]
    >>> gather_lists(list_)
    [1, -1, 0, 2, 3, 4]
    """
    
    # special form of sum for "adding" lists
```
```python
def gather_lists(list_: Union[List, object]) -> List[object]:
    """
Return the concatenation of the sublists of list_.

>>> list_ = [[1, 2], [3, 4]]
>>> gather_lists(list_)
[1, 2, 3, 4]
>>> list_ = [[1, [-1,0],2], [3, 4]]
>>> gather_lists(list_)
[1, -1, 0, 2, 3, 4]
    """

# special form of sum for "adding" lists
return sum(([gather_lists1(x) for x in list_], [])
    if isinstance(list_, list)
    else [list_]
```
def list_over(obj: Union[list, str], n: int) -> List[str]:
    """
    Return a list of strings of length greater than n in obj,
    or sublists of obj, if obj is a list.
    If obj is a string of length greater than n, return a list
    containing obj. Otherwise return an empty list.
    """

    >>> list_over("five", 3)
    ['five']
    >>> list_over("five", 4)
    []
    >>> list_over(["one", "two", "three", "four"], 3)
    ['three', 'four']
    """
```python
def list_over(obj: Union[list, str], n: int) -> List[str]:
    """
    Return a list of strings of length greater than n in obj, or sublists of obj, if obj is a list.
    If obj is a string of length greater than n, return a list containing obj. Otherwise return an empty list.
    """

    if isinstance(obj, list):
        return sum([[list_over(x, n) for x in obj], []])
    else:
        return [obj] if len(obj) > n else []
```
def list_even(obj: Union[list, int]) -> List[int]:
    
    """
    Return a list of all even integers in obj, or sublists of obj, if obj is a list. If obj is an even integer, return a list containing obj. Otherwise return an empty list.
    """

>>> list_even(3)
[]

>>> list_even(16)
[16]

>>> list_even([1, 2, 3, 4, 5])
[2, 4]

>>> list_even([1, 2, [3, 4], 5])
[2, 4]

>>> list_even([1, [2, [3, 4]], 5])
[2, 4]
"""
def list_even(obj: Union[list, int]) -> List[int]:
    
    """Return a list of all even integers in obj, or sublists of obj, if obj is a list. If obj is an even integer, return a list containing obj. Otherwise return an empty list."

>>> list_even(3)
[]
>>> list_even(16)
[16]
>>> list_even([1, 2, 3, 4, 5])
[2, 4]
>>> list_even([1, [2, [3, 4]], 5])
[2, 4]
    """

if isinstance(obj, list):
    return sum([list_even(x) for x in obj], [])
else:
    return [obj] if obj%2==0 else []
def count_all(obj: Union[list, object]) -> int:
    """
    Return the number of elements in obj or sublists of obj
    if obj is a list.
    Otherwise, if obj is a non-list return 1.
    """

    >>> count_all(17)
    1
    >>> count_all([17, 17, 5])
    3
    >>> count_all([17, [17, 5], 3])
    4
    """
```python
def count_all(obj: Union[list, object]) -> int:
    ""
    Return the number of elements in obj or sublists of obj
    if obj is a list.
    Otherwise, if obj is a non-list return 1.
    
    >>> count_all(17)
    1
    >>> count_all([17, 17, 5])
    3
    >>> count_all([17, [17, 5], 3])
    4
    ""
    if isinstance(obj, list):
        return sum([count_all(x) for x in obj])
    else:
        return 1
```
To save space, let us denote: \texttt{count\_all} as \texttt{ca} so that \texttt{count\_all([1,2] is \texttt{ca}([1,2]))}

\[
\begin{aligned}
\texttt{ca([5, , 4, 6])} \\
\texttt{ca([3, , 1])} \\
\texttt{ca([1,2, ,3])} \\
\texttt{ca([ca(2)])}
\end{aligned}
\]

\[
\begin{aligned}
\texttt{sum([1, 6, 1, 1])} \\
\texttt{sum([1, 4, 1])} \\
\texttt{sum([1, 1, 1, 1])} \\
\texttt{sum([1])}
\end{aligned}
\]

\[
\begin{aligned}
>>> \texttt{ca([5, [3, [1,2,[2,3], 1], 4 , 6])}
9 \# answer is 9
\end{aligned}
\]
def count_even(obj: Union[list, int]) -> int:
    
    Return the number of even numbers in obj or sublists of obj if obj is a list. Otherwise, if obj is a number, return 1 if it is an even number and 0 if it is an odd number.

    >>> count_even(3)
    0
    >>> count_even(16)
    1
    >>> count_even([1, 2, [3, 4], 5])
    2

    """
```python
def count_even(obj: Union[list, int]) -> int:
    """
    Return the number of even numbers in obj or sublists of obj
    if obj is a list. Otherwise, if obj is a number, return 1
    if it is an even number and 0 if it is an odd number.
    """

    >>> count_even(3)
    0
    >>> count_even(16)
    1
    >>> count_even([1, 2, [3, 4], 5])
    2
    """

    if isinstance(obj, list):
        return sum([count_even(x) for x in obj])
    else:
        return 1 if obj % 2 == 0 else 0
```

def count_above(obj: Union[list, int], n: int) -> int:
    """
    Return tally of numbers in obj, and sublists of obj, that are over n, if
    obj is a list. Otherwise, if obj is a number over n, return 1. Otherwise
    return 0.
    """
    >>> count_above(17, 19)
    0
    >>> count_above(19, 17)
    1
    >>> count_above([17, 18, 19, 20], 18)
    2
    >>> count_above([17, 18, [19, 20]], 18)
    2
    """
def count_above(obj: Union[list, int], n: int) -> int:
    
    ""
    Return tally of numbers in obj, and sublists of obj, that are over n, if obj is a list. Otherwise, if obj is a number over n, return 1. Otherwise return 0.
    return 0.

    >>> count_above(17, 19)
    0
    >>> count_above(19, 17)
    1
    >>> count_above([17, 18, [19, 20]], 18)
    2
    ""
    
    if isinstance(obj, list):
        return sum([count_above(x, n) for x in obj])
    else:
        return 1 if obj>n else 0
```python
def max_length(obj: Union[list, object]) -> int:
    
    """
    Return the maximum length of obj or any of its sublists, if obj is a list.
    otherwise return 0.
    """

    >>> max_length(17)
    0
    >>> max_length([1, 2, 3, 17])
    4
    >>> max_length([1,[[1, 2, 3,2, 3,2, 3,2, 3], 4, [4, 5]]])
    9
    """
```
```python
def max_length(obj: Union[list, object]) -> int:
    """
    Return the maximum length of obj or any of its sublists,
    if obj is a list.
    otherwise return 0.
    """
    if isinstance(obj, list):
        t = [max_length(x) for x in obj]
        return max(t) if max(t) > len(t) else len(t)
    else:
        return 0
```