Lecture (9/2)
Statement - executed by the interpreter to perform an action

```
Statement
        Clause
        <header>:
                Suite
                <statement>
                <statement>

        Clause
        <header>:
                Suite
                <statement>
                <statement>
```

False values in Python: False, 0, ‘ ’, None
True values in Python: Anything else

Lecture (9/4)
Same line is executed at the same time
Right side of assignment is computed before the left side is assigned

Function’s *domain* is the set of all inputs it might possibly take as arguments
Function’s *range* is the set of output values it might possibly return

Guide to designing Function
- Give each function exactly one job, but make it apply to many related situations
- Don’t repeat yourself (DRY): Implement a process just once, but execute it many times

Generalizing Patterns with Arguments
- Assert: must be true or give an error
- Higher order function: function that takes a function as a parameter
    - Can use def name as a parameter

Locally defined functions
- Functions defined within

Lecture (9/6): Environments for Higher-Order Functions

Environments for Higher-Order Functions

Functions are first-class: they are values in Python, used like numbers

Higher-order function: A function that takes a function as an argument value or returns a function as a return value

- Different between calling function and passing a function through
- Every user-defined functin has a parent frame, every parent is created when it is defined

How to Draw a Environment Diagram

1. Add a local frame, titled with the <name> of the function being called

Lambda

- expression that evaluates to a function
- No return keyword
- 1 line body

- Only def statement gives the function an intrinsic name or else it
- Name of lambda function is just lambda(x) <line 1>

Lecture (9/9)

Return Statement

A return statement completes the evaluation of a call expression and provides its value:

- After return, switch back to the previous environment; f(x) now has a value
- If you don't have a return statement, it defaults to return None

Self Reference

- Returning the same function

Control

Execution Rule for Conditional Statements:

1. Evaluate th eheader's expression
2. If it is a true value (or an else header), execute the suite & skip the remaining clauses

Evaluation Rule for Call expressions:

1. Evaluate the operator and then the operand subexpressions
2. Apply the function that is the value of the operator to the arguments

- Control helps prevent errors by setting where it can run

Control Expressions

- Logical Operators
    - To evaluate <left> and <right>:
    1. Evaluate the subexpression <left>

2. If the result if false, expression evaluates to false
- To evaluate <left> or <right>:
1. Evaluate the subexpression <left>
2. If the result t

Lecture 7 (9/11)

Functional Abstraction

What do you need to know about functions?

- Argument "one argument"
- Square computes the square of a number

Don't need to know

- Intrinsic name "square"
- Square computes the square by calling mul

Choosing Names

- Don't matter for correctness **but** matter for composition
- Type of value bound to the name is best documented in a function's docstring

Which Values deserve a Name

- Repeated compound expressions
- Meaningful parts of complex expressions
- Names can be long if they help document your code
- Short represent quanities
    - n, k, i - integers
    - x, y, z - real numbers
    - f, g, h - functions

WAV Files

- Waveform Audio File Format encodes a sampled sound wave
-

Lecture 8 (9/13)

Currying

- Better to know what each part of the function does

Function Decorators

- Traced function that is a higher order function that prints
- @trace
- @ calls a higher order function on the function you just defined
    - @<func name>
- Triple = trace1(tripe)
- Function decorator - @trace1
- Decorated function - function after

Review

WWPD

- Print function returns None, it displays its arguments (separated by spaces) when it is called
- Displays it unless it is None
- First evaluate the inside parameter call

- Names in nested def statements can refer to their enclosing scope, in other words it can use the variables inside the outer function
- Describe the function instead of environment diagram

Environment Diagrams

- When creating a def statement
    - Create function
    - Bind name of function to function value

Lecture 9 (9/18)

Recursion

Def: A function is called recursive if the body of that function calls itself, either directly or indirectly

Ex.) Sum of the digits of 2019 = 12

We can break the problem of summing the digits of 2019 into *a smaller instance of the same problem*

Anatomy of a Recursive Function

- Conditional statements check for base cases
- Base cases are evaluated without recursive calls
- Recursive cases are evaluated with recursive calls

- Different frames keep track of athe different arguments in each call
- What n evaluates to depends upon the current environment

Iteration vs Recursion

Iteration is a special case of recursion

Can write any iteration with recursion

Iteration to Recursion

Idea: The state of a iteration can be passed as arguments

Iteration: Pass through assignment statements that change itself

Recursion: pass through as argument

Verifying Recursive Functions: The Recursive Leap of Faith

Is fact implemented correctly?

1. Verify the base case
2. Treat fact as a functional abstraction

3. Assume fact(n-1) is correct
4. Verify that fact(n) is correct

<u>Mutual Recursion: The Luhn Algorithm</u>

- Used to verify credit card numbers
-

Lecture 10 (9/20) Tree Recursion

Tree Recursion

<u>Order of Recursive calls</u>

Inverse Cascade

- Where recursive function is called matters

<u>Tree Recursion</u>

- Tree-shaped processes arise whenever executing the body of a recursive function makes more than one recursive call

Lecture 11 (9/23) Lists

Lists

- Lists
    - [<item>, <item>]
- len(<list>)
    - Gives the number of elements

<u>Operations</u>

- <list> * 2
    - Gives the same list twice
    - Multiplying by 0 gives an emply list
- <list> + <list>
    - Gives one list with both elements
- Nested Lists
    - [<list>, <list>]

<u>Containers</u>

- Built in operators for testing whether an element appears in a compound value
- Digits = [1,3,3]
    - 1 **in** digits
    - 1 **not in** digits

<u>For Condition</u>

- **Sequence Iteration**
- For <name> in <expression>:
    - Evaluate the header expression which must be iterable
    - **For** elem **in** s:
        - Assigns name to each element

- Sequence Unpacking in For Statements
    - For x, y in pairs:
    - [[1 , 2] , [3 , 4]]
    - it will unpack the two numbers in pairs
    - Zip makes two lists into pairs
- **Range Type**
- Range is a sequence of consecutive integers
    - range(-2,2) does not include the last element
    - Class range, is not a list
- List function converts into a list or list constructor
- For i in range(n):
- Can use underscore when you dont use the iterator
    - For _ in range(3):

<map exp> for <name> in <iter exp> if <condition>

Lecture 12 (9/25) Data Abstraction

Data Abstraction

- A methodology b which functions enforce an abstraction barrier between **representation** and **use**
- Constructors - returns function
- Selectors - returns number
- Unpacking a list
    - X,y = [1,2]

What are Data?

- Guarantee that constructor and selector functions work together to specify the right behavior
- Data abstraction uses selectors and constructors to define behaviors

Lecture 13 (9/27)

Box-and-Pointer Notation
Closure Property of Data Types
A method for combining data values satisfies the closure propterty if
- The result can be

Lists are represented as a row of index-labeled adjacent boxes, one per element
Each box either contain a primitive value or points to a compound value

Slicing
<list>[start:end]
Slicing creates new values, not references

Processing Container Values
Sequence Aggregation
Max or min(interable, key= func)
sum(iterable, start = num)
all(iterable, if everything is true)

Trees
Recursive description (wooden tree):
- A tree has a root label and a list of branches
- Each branch is a tree
- A tree with zero branches is called a leaf
- A tree starts at the root

Relative description (family tree):
- Each location in a tree is called a node
- Each node has a label that can be any value
- One node can be the parent/child of another
- “Each parent is the sum of its children”
- The top node is the root node

Path: a sequence of nodes, path of a root to a leaf

Implementing the Tree Abstraction

Lecture 14 (9/30)
Objects
Objects represent information with data and behavior to create abstractions
A type of object is called a class, classes are first-class values in Python
Object oriented programming:
- A metaphor for organizing large programs
- Special syntax that can improve the composition of program (ex: .)

In Python, every value is an object
- All objects have attributes
- Happens through object methods
- Functions do one thing; objects do many related things

Representing Strings: the ASCII Standard
Representing Strings: the Unicode Standard
- All characters from all languages

Mutation Operations
Objects can change states, but not numbers
- Only objects of *mutable types* can change: lists & dictionaries

Tuples
Uses parentheses
- Can't change values of tuples, unmutable
- Can use them as keys in a dictionary
- A list inside a tuple can still change

Identity Operators
<exp0> is <exp1>
Eval to True if both <expo0> and <exp1> are the same object

Lecture 15 (10/2)

Mutable Functions
Nonlocal - sets variable to the nonlocal frame, rebind variable to the first non-local frame in which it was bound previously
Effect: Future assignments to that name change its pre-existing binding in the first non-local frame of the current environment
nonlocal < variable name>
If you don't' set nonlocal, it would create a new variable with the same name

Python Particulars
- Precomputes which frame contains each name before executing the body of a function
- Within the body of a func, all instances of a name must refer to the same frame.
    - May give an error if you use both in one frame

Referential Transparency
If you change a method to a number, it will still stay the same answer
Not transparent means that it matters how many times the function is called

Lecture 16 (10/4)

Iterators

A container can provide an iterator that provides access to its elements in order
Methods - iter(iterable): return an iterator over the elements or an iterable value
    next(iterator): return the next element in an iterator
EX)
```
S = [3,4,5]
 t = iter(s)
>>>next(t)
3
>>>next(t)
4
```

View of a Dictionary
An *iterable* value is any value that can be passed to iter to produce an iterator
An *iterator* is returned from iter and can be passed to the next; all iterators are mutable

Built in functions for iteration
map(func, iterable): Iterator over func(x) for x in iterable
filter(func, iterable): Iterator over x in iterable if func(x)
zip(first_iter, second_iter): Iterate over co-indexed (x,y) pairs
reversed(sequence): Iterate over x in a sequence in reverse order

Lazy functions - don't compute until you need the elements, stores the elements in an object
- Doesn't call function on x until you call next(m) or list(m)

To view the contents of an iterator, place the resulting elements into a container
List (iterable)
Tuple (iterable)
sorted(iterable)

Generator
Yield keyword
- Can yield multiple times

If a function has yields, it returns a generator which is an iterator
Generator function is a function that *yields* values instead of returning them
Normal function *returns* once, generator function can yield multiple times
A generator is an iterator created automatically
It returns a generator object

Yield from statement yields all values from an iterator or iterable

Can do recursive

Lecture 17 (10/7)

Objects

Object Oriented Programming : A method for organizing programs

- Data Abstraction
- Bundling together info and related behavior

A metaphor for computation using distributed state

Method calls are messages passed between objects

Several objects may all be instances of a common type

Classes

Describes the general behavior of its instances

All bank accounts should have withdraw and deposit behaviors that all work in the same way

```
a = Account('John')
>>>A.holder
John
>>>A.balance
0
```

```
class <name>:
        <suite>
```

Suite is executed when the class statement is executed

Object Construction

A = account('Jim')

A new instance of that class is created:

The __init__ method of the class is called with

```
Class Account:
        Def __init__(self, account_holder):
                Self.balance = 0
                Self.holder = account_holder
        Def deposit(self, amount):
                Self.balance = self.balance + amount
                Return self.balance
        Define withdraw(self, amount):
                If amount > self.balance:
                        Return "insufficient funds"
                Self.balance -= amount
                Return self.balance
```

Every object that is an instance of a user-defined class has a unique identity

Is and *is not* tests if two expressions evaluate to the same object

Methods

- Methods are functions defined in the suite of a class statement
- All invoked methods have access to the object via the self parameter, so they can all access and manipulate the object's state
- Instance before the is bound to self and the method is invoked with 1 parameter

Dot Expressions

- Objects receive messages via dot notation
- Dot notation accesses attributes of the instance or its class
    - <expression>.<name>
- Can be any Python expression
- Name must be a simple anem
- Value of the attribute looked up by <name> in the object that is the value of the <expression>
- Dot expression then call expression
- getattr(tom_account, "balance")
- hasattr(object, "name")
- Looking up an attribute name in an object may return:
    - One of this instance attributes, or one of the attributes of its class

Methods and Functions

Python distinguishes between

- Functions, which we have been creating since the beginning of the course,and
- Bound methods, which couple together a function and the object on which that method will be invoked

1. Evaluate the <expression> to the left of the dot, which yields the object of the dot expression
2. <name> is matches against eh instance attributes of that object; if an attribute with that name exists, its value is returned
3. If not, <name> is looked up in the class, which yields a class attribute value
4. That value is returned unless it is a function, in which

Class attributes are "shared across all instance of a class bc they are attributes of the class, not the instances

    - interest

Lecture 20 (10/14)

Linked Lists

Linked list is either empty or a first value and the rest of the linked list

Lecture 21 (10/16)

Efficiency
How many calls
Memoization

- Remember the results that have been computed before

```
Def memo(f)
        Cache = {}
        Def memoized(n):
                If n not in cache:
                        Cache[n] = f(n)
                Return cache[n]
        Return memoized
```

Exponentiation
Orders of Growth

- Quadratic Time
    - Incrementing n increases time by n times a constant
    - Ex. nested for loop
- Exponential Time
    - Incrementing n multiplies time by a constant
    - Ex. Tree recursion without memoization
- Linear growth
    - Incrementing n increases time by a constant
    - Ex. exp
- Logarithmic growth
    - Doubling n only increments time by a constant
    - Ex. exp_fast
- Constant growth
    - Increasing n doesn't affect time
    - Ex. looking up a key in a dictionary

Space

- Which environment frames do we need to keep during evaluation?
- At any moment there is a set of active environments
- Values and frames in active environments consume memory
- Memory that is used for other values and frames can be recycled
- Active environments:
    - Environments for any function calls currently being evaluated
    - Parent environments of functions named
- Fib has exponential time but linear space,

Lecture 22 (10/18)

Modular Design
Separation of Concerns

- A design principle: Isolate different parts of a program that address different concerns
- Modular component can be developed and tested independently

-
-

Lecture 23 (10/21)

Data Examples

Lists in Environment Diagrams
Append: reference to t
Extend: copies numbers in t to s
list(s) creates a new s
Slice assignment: will replace elements, not a reference, could shift the elements down
Remove: takes out the first element equal to elem

Lecture 25 (10/30) Handling errors

Exceptions: Raises an exception whenever an error occurs
Exceptions can be handled by the program, preventing the interpreter from being halting
Unhandled exceptions will cause Python to halt execution and print a stack trace
Mastering exceptions:

- Exceptions are objects, they have classes with constructors
- They enable non-local continuation of control

Assert statements raise an exception of type AssertionError
If you run python with python3 -0 it will ignore asserts

Raise Statements

- Raise <expression>

TypeError
NameError
KeyError
RecursionError

Try

```
Try:
        <try suite>
Except <exception class> as <name>:
        <except suite>
```

Jumps to the except suite of the most recent that matches the type

Interpreters

Eval

Base Cases:

- Primitive values (numbers)
- Look up Values bound to symbols

Recursive calls:

- Eval (operator, operands) of call expressions
- Apply (procedure, arguments)
- Eval (sub-expressions) of special forms

Apply

Base Cases:

- Built-in primitive procedures

Recursive calls:

- Eval (body) of user-defined procedures

Special forms are identified by the first element
Any combination that is not a known special form is a call expression

Frames and Environments

- A frame represents an environment by having a parent frame
- Frames are Python instances with define and lookup function

Define

- Define binds a symbol to a value in the first frame of the current environment

Lecture 29 (11/6) Tail Calls

Dynamic scope - The parent of a frame is the environment in which a procedure was called
Lexical scope - The parent of a frame is the environment in which a procedure was defined

Advantages of functional programming

- The value of an expression is independent of the order in which sub-expressions are evaluated
- Sub-expressions can safely be evaluated in parallel or only on demand
- Referential transparency: the value of an expression does not change when we substitute one of its subexpressions with the value of that subexpression

Recursive calls always create new active frames
Nothing left to do after tail call

A tail call is a call expression in a tail context:
- The last body sub-expression in a lambda expression (or procedure definition)
- Sub-expression 2&3 in a tail context if expression
- All non-predicate sub-expression in a tail context cond

Lecture 30 (11/8)

Macro is an operation performed on the source code of a program before evaluation
Define-macro special form that defines a source code transformation
Evaluation procedure of a macro call expression~
- Evaluate the operator sub-expression, which evaluates to a macro
- Call the macro procedure on the operand expressions without evaluating them first
- Evaluate the expression returned from the macro procedure

Lecture 31 (11/13)

Sequence Operations
Streams are lazy Scheme Lists
- List but the rest of the list is computed only when needed
- (car (cons 1 nil)) > 1 (car (cons-stream 1 nil)) > 1
- Cons-stream 1 (cons-stream (/ 1 0) nil)) > 1 not error
- Only computes the car when needed, but will not evaluate the cdr

Lecture 32 (11/15)

Database Management Systems (DBMS)
- A table is a collection of records, which are rows that have a value for each column
- A column has a name and a type
- A row has a value for each column
- Structured Query Language (SQL) declarative language

Declarative languages (SQL & Prolog)
- A program describes the desired result
- **Imperative language** (Python & Scheme)
    - A program is a description of computational processes
    - Carries out execution/evaluation rules

SQL Overview
- *Select* statement creates a new table, either from scratch or by projecting a table
- *Create table* statement gives a global name to a table

Selecting Value literals
- *Select* statement always includes a comma-separated list of column descriptions
    - Result of a SELECT statement is displayed to the user, but not stored
- ex) select [expression] as [name], [expression] as [name];
- ex) create table [name] as [select statement];

- Selecting literals creates a one-row table
- The union of two select statements is a table containing the rows of both of their results
- Select statement can specify an input table using a from clause

Select Statements Project Existing Tables

- ex) select [columns] from [table] where [condition] order by [order];
- If you want all the columns then use *

Arithmetic in Select Expressions

- ex) *select* chair, single + 2 * couple as total from lift;
- ex) select word, one + two + four + eight as value from ints order by value;
- ex) select word from ints where one+ two/2 + four/4 + eight/8 = 1 order by one + two + four + eight;

<u>Lecture 33 (11/18)</u>

Select [columns] From [table] where [condition] order by [order];
Join two table rows by using comma

<u>Lecture 34 (11/20)</u>

Aggregate Functions
Gives one output row
Select [columns] from [table] group by [expression] having [expression]