Welcome to CS61A Disc 1 Sect. 29/47 :D

Dickson Tsai
dickson.tsai@berkeley.edu
OH: Tu, Th 4-5pm 411 Soda

Previous: Working Effectively in CS **>>>**

Today: Statements/Expressions/ Functions >>>

Next stop: Higher Order Functions

Anuncios

- Labs are generally due Wednesdays at 11:59PM
- Section homework: for me to help track your progress/ attendance. More info at the end.
- To get to my website:
  - dicksontsai.com/cs61a OR:
    - Go to cs61a.org
    - Click the "Staff" link and click my name
    - Go to Discussion
  - You will find my slides to discussion/lab. They may be different, because I add more content along the day.

Programming Language

- Programming involves acquiring a language!!!
  - How many programs are possible? How many sentences are possible?
  - We don't memorize all the sentences that exist. There are infinitely many! The house next to the house next to …
  - We combine parts together. The way we combine them is also meaningful.
  - Programming language: **primitive** expressions, means of **combination**, means of **abstraction**
  - [Discuss] Can you think of examples of primitive expressions, means of combination, and means of abstraction in English?

Programming Language

- Programming language: **primitive** expressions, means of **combination**, means of **abstraction**
  - Previous question: Can you think of examples in English?
  - *Primitive expressions* = words. E.g. walk, run, red, Bob
    - Some words belong to certain categories, e.g. verb, adjective.
    - Similarly, there are different data types: e.g. string, integer
  - *Means of combination* = syntax. E.g. You are hardworking.
    - Call exps allow nesting
  - *Means of abstraction* = abbreviations, e.g. afaik, idk, laser, Harry Potter (name for 500 pages of text), "that lecture"
    - So far we've encountered names and functions

Programming Language

- What do both types of languages try to achieve?
  - We want to *express* complex ideas (natural language) and complex computations (programming language), but do so easily.

Defining Functions + Warmup

- Recall how function definition works
- Try out the warmup.
- [Discuss] Confirm with your neighbors that you have the right answer.
  - If your answers do not match, don't just copy your partner's answer. See if you can modify your work to get your partner's answer. Writing your thought process down helps.

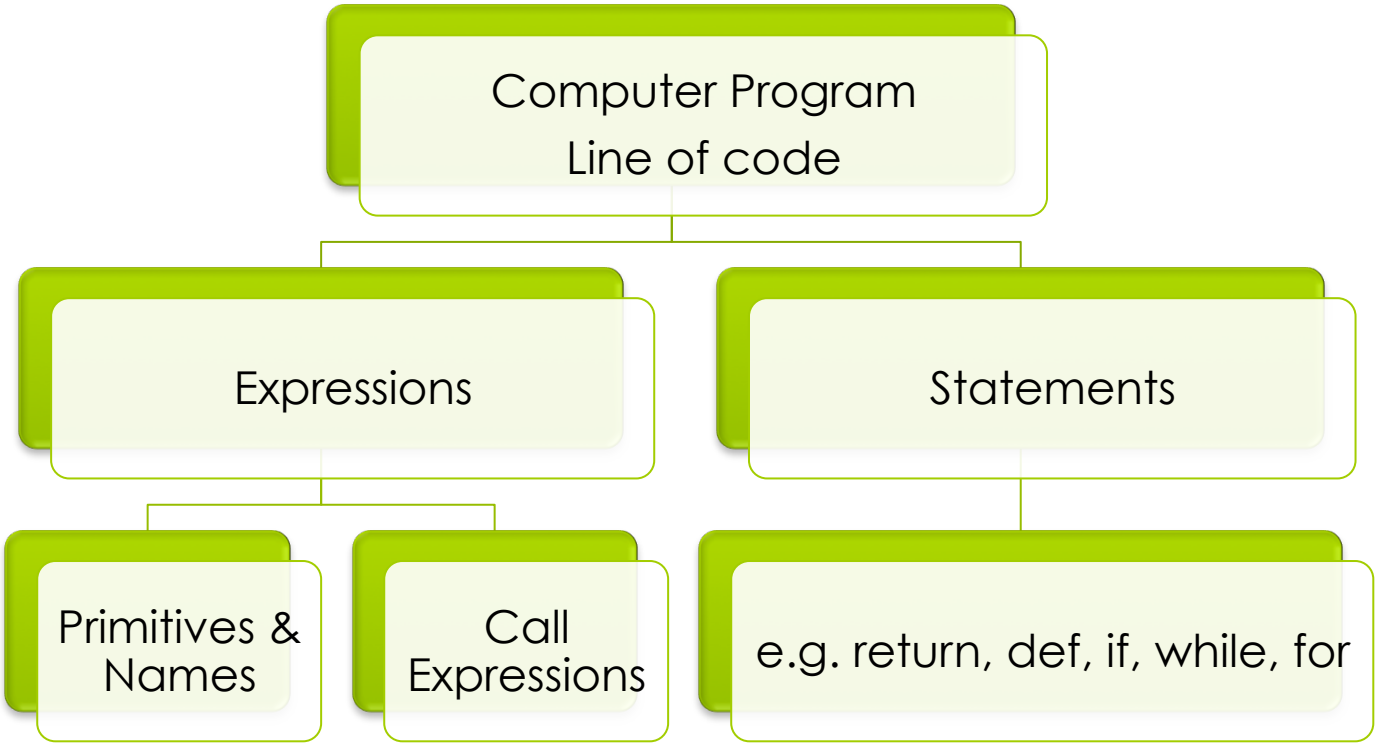
Defining Functions + Warmup

- Recall how function definition works:

```
def fn(x):
    return 2 ** x
fn(4)
```

```
def foo():
    return bar
bar = 2
foo()
```

From the reading



Each type of code has its own procedure. E.g. names have a lookup procedure, assignment has rules

Expression

- An expression describes a computation and evaluates to a value.
- Computation – process that transforms input to output
  - E.g. square(4) describes the process of squaring 4, whose eventual value should be 16.
  - If everything works correctly, square(4) is a convenient abstraction for the code that actually does the squaring
- Evaluate - convert expression to value (expression -> value)
  - It's like a function. There are rules you have to follow for each input.

Evaluation Rules

- Primitive Expressions
  - A number evaluates to a number
  - A string evaluates to a string
  - Primitive expressions evaluate to themselves!!
- Names – Lookup Procedure
  - Look in the current frame
  - If not found, go look in the parent frames (the frame where the function is defined).
- Call Expression
  - Evaluate the operator and then the operands (from L to R)
  - Apply the function (value of the operator) to the arguments (values of the operands)

Pop Quiz

- Q: What does the value of the operator have to be?
- A: A function

- Q: What expression can the operator be?
- A: An operator can be any expression whose value is a function. It can be a name, call expression, etc.

Statement

- A statement is *executed* by the interpreter to achieve an effect.
- Each statement has its own rules! You just have to memorize them
- E.g. assignment, def, while, for, if
- Suites are indented (e.g. this block of code belongs to this statement). All the lines belong to the function.

```
def print_lots():
        print(1)
        print(True)
        print("hello world")
```

Pure vs Non-pure function

- Whenever an expression is typed in directly to the interpreter, the interpreter prints it out. Why?
- A: e.g. You can use Python like a calculator!

```
>>> 1 + 1          vs.          >>> print(1 + 1)
2                                2
```

- The printing is different for some values!

```
>>> None           vs.          >>> print(None)
>>>                              None
```

The print function is non-pure because it has this side effect.

Recall: Rule for Evaluating Call Exps

- Evaluate the operator, then the operands (from L to R)
- *Apply* the function (value of the operator) to the arguments (values of the operands)
  - But what does apply mean?
  1. Create a new frame. This will be the function call's own namespace (closure)
  2. Bind the formal parameters to the arguments
  3. Evaluate the body with the new frame as the current frame.

Recall: Rule for Evaluating Call Exps

- Evaluate
- *Apply* the function (value of the operator) to the arguments (values of the operands)
  1. Create a new frame.
  2. Bind the formal parameters to the arguments
  3. Evaluate the body with the new frame as the current frame.

```
def square(x):
    return x * x
square(4)
```

[Demo]

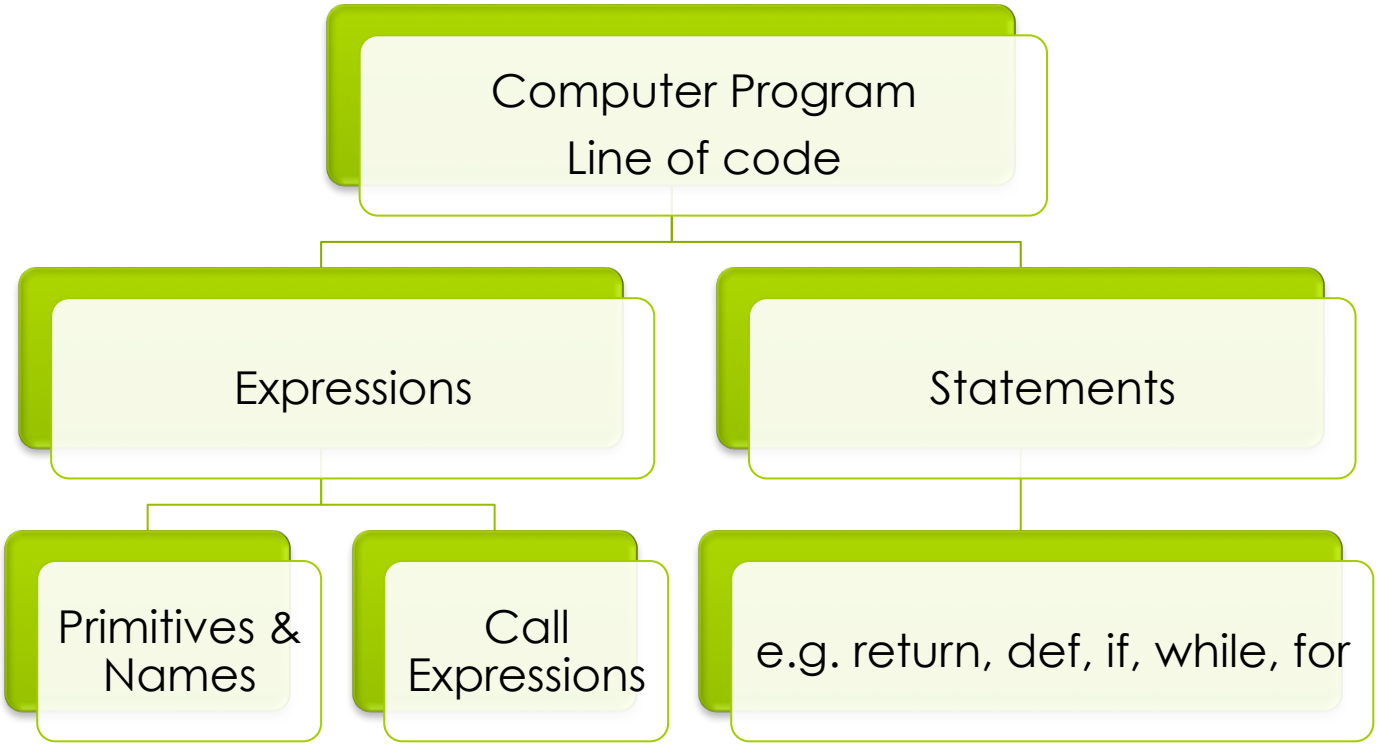
Recall: Other Important Rules

- Assignment
  - Evaluate all of the expressions on the RHS of the assignment operator (the =), from left to right
  - Bind all the names on the LHS to the resulting values in the current frame
  - [Tip] Names can only bind to values, not other names
- Lookup
  - Look in the current frame
  - If not found, go look in the parent frames (the frame where the function is defined).
- You do not have to display built-in functions/func calls

Common Mistakes

- Names have to be looked up, starting from the current frame. Do not make a guess based on intuition.
- The `return` statement is important for a function:
  - Immediately exits the body of the function and 'returns' control to the caller.
  - The resulting value of a call expression will be the same as the resulting value next to the `return` statement
  - If no return is present, Python will return None by default at end of function
- Try >>> "hi" vs. >>> print("hi"). What's the difference?
- Interpreter printing (vs. using print function) only happens in interactive mode. (i.e. when there's >>>)
- The value of a call to print() is always None.
- Please trust the rules! Memorize them and be confident in them ☺

In Summary


Computer Program
Line of code
Expressions
Statements
Primitives & Names
Call Expressions
e.g. return, def, if, while, for

Section Homework

- This will be used to ***mark attendance*** for the next section for midterm recovery. Will help participation (but you can participate in other ways as well!).
- Some other TAs give quizzes instead.
- These should take at most 15 minutes, if you keep up with your reading.
- *Submit using any format (email/next week)*. If email, please use dickson.tsai+cs61ahw@berkeley.edu .
- Graded on effort, but use your time wisely. 10 minutes of BS < 10 minutes of ***citing*** the textbook

Section Homework

We're going to write a function that takes in two numbers and returns the sum of their square roots.

1. You can break this problem down: how to define a function, how to add, how to take a square root. Google if you are stuck.

1. What will the function signature be?
2. Before you code, write a docstring for your function with doctests. Given 25 and 16, what do you expect the output of this function to be?
3. Write the code for this function.
4. Draw an environment diagram for the code to the right. Replace "your_fn" with the function above. Your name for the function must be better than "your_fn"
5. How can I improve your discussion experience?

```
def your_fn…:
        …
a, b, c = max, your_fn, \
your_fn(9, 9)

b(a(c, your_fn(16, 25)),\
your_fn(c + 3, 6 * 6))
```

Additional Problems (not collected) Credit: Andrew Huang

1. What would Python print?

```
>>> 3 + 4

>>> '3' + 4

>>> 1 / 0

>>> def boom():
...     return 1 / 0
...

>>> boom()
```

Additional Problems (not collected) Credit: Andrew Huang

```
>>> def square(x):
…       return x * x
…
>>> def so_slow(num):
…       return num
…       num / 0
…
>>> square(so_slow(5)) #What's the output? Why?
```

Additional Problems (not collected) Credit: Andrew Huang

```
>>> def foo(x):
…       print(x)
…       return x + 4
…
>>> def bar(x):
…       x
…       return x + 4
…
```

What is the difference between foo and bar?

Additional Problems (not collected) Credit: Sumukh Sridhara

What gets printed?

```
>>> def hi(x)
…       print('x')
…       return 42
…       print('Oreos')
…

>>> hi(0)
```