Lecture #9: Still More on Functions

Announcements

- Sign up for advising (Piazza @706).
- Practice Midterm Tuesday from 5-7PM. Do be sure to check the practice exam out when you have time. It will be based on the Fa20 first test.
- Drop deadline coming up: 10 Feb (Wednesday).
- If you want the unit for CSM (Computer Science Mentors), you'll need to get it soon before the add deadline. Lots of mentoring spots still available (Piazza @580).
- Please submit exam conflict forms by Thursday (see Piazza @318).
- Ask questions on the Piazza thread for today's lecture (Piazza @719).

Exercise: Reversing Digits

- Problem: I want a function that reverses the digits in a number.
- For example, I'd like to have
  reverse_digits(1234) == 4321

Exercise: Interleaving Digits

- Problem: I want a function that, given two numbers, $A$ and $B$, containing the same number of digits, returns the result of interleaving the digits of $A$ and $B$, starting with the first digit $A$, then the first digit of $B$, then the second digit of $A$, etc.
- For example, I'd like to have
  interleave_digits(13579, 24680) == 1234567890
def flip(flop):
    if ______:
        flip = ______
        return flip

def flop(flip):
    return flop

flip, flop = flop, flip
flip(flip(1))  (3)
def flip(flop):
    if flop == 3:
        flip = lambda flip: 3
    return flip

def flop(flip):
    return flip

flip, flop = flop, flip
flip(flop(1)(2))(3)

---

def flip(flop):
    if flop == 3:
        return None
    flip = lambda flip: 3
    return flip

def flop(flip):
    return flip

flip, flop = flop, flip
flip(flop(1)(2))(3)

See this in the Python Tutor

---

Exercise: Tracing

- I'd like a function `trace1` that takes as its argument a one-argument function (say `f`) and returns a one-argument function that
  - Prints its argument, preceded by a `->`,
  - Prints the value of `f` applied to its argument, preceded by a `<-`,
  and then returns that value.

- So,

  ```python
  >>> def square(x):
  ...     return x*2
  ...  
  >>> square(3) + square(4)
  -> 3
  <- 9
  -> 4
  <- 16
  25
  ```
Decorators

- Python has an interesting feature—**decorators**—that exploits higher-order functions in a useful way.

- The notation
  ```python
  @ATTR
def aFunc(...):
  ...
  ``
  where ATTR is some expression, is essentially equivalent to
  ```python
  def aFunc(...):
  ...
  aFunc = ATTR(aFunc)
  ```

- So, having defined `trace1`, we can now write
  ```python
  @trace1
def square(x):
  return x * x
  ```
  and see
  ```
  >>> x = square(4)
  ~> 4
  <= 16
  >>> x
  16
  ```

Why Do It That Way?

- What's wrong with this alternative way to trace?
  ```python
  def aFunc1(x):
  ...
  aFunc = trace1(aFunc1)
  ```

- Consider
  ```python
  def fib1(n):
  return 0 if n <= 0 else 1 if n == 1 else fib1(n-2) + fib1(n-1)
  fib = trace1(fib1)
  ```
  A call such as `fib(4)` will trace only the outer call, not the recursive inner calls.