Computer Science I — CSCI141
Fruitful Functions — Lecture (2/2)

08/26/2013

1 Fruitful Functions

Many of the functions we've encountered so far can be understood as commands; however there is another important class of functions that yield results. We call this class *fruitful functions*. For example, `turtle.forward` doesn't yield a result; it moves the turtle. But the function `math.sqrt` does yield a result.

```
>>> turtle.forward(100)
>>> math.sqrt(2)
1.4142135623730951
```

We can also write fruitful functions. To do so we need to know how to tell Python that our function should yield a result. The special word to use is `return`, which causes the function to exit yielding the value specified.

Example

```
def addOne(x):
    """addOne: Number -> Number"""
    return x+1

>>> addOne(2)
3
```

2 Two Classic Fruitful Functions

There are two mathematical functions that computer scientists like to use as examples: the factorial function and the Fibonacci function. The factorial of a natural number n is written as $n!$, the nth Fibonacci number is written F_n. The mathematical definitions are below.

$$\begin{aligned} 0! &= 1 \\ n! &= n \times (n-1)! \end{aligned} \qquad \begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_n &= F_{n-1} + F_{n-2} \end{aligned}$$

The translations of these functions to code now follows.

```python
def fact(n):
    """fact: NatNum -> NatNum"""
    if n == 0:
        return 1
    else:
        return n * fact(n-1)

def fib(n):
    """fib: NatNum -> NatNum"""
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

3 Substitution Trace

The technique of constructing execution diagrams is important and useful for understanding and debugging of recursive functions. However, it involves a lot of details that can be omitted for a certain classes of functions. Another form of tracing called *substitution tracing* can eliminate a lot of that detail when tracing fruitful functions.

Substitution tracing is a generalization of arithmetic expression evaluation. It involves writing a function call with its arguments, an equal sign, and then the expression it evaluates to. This process is repeated until the result is computed.

3.1 Example: Factorial

$$\begin{aligned}
\texttt{fact(3)} &= 3 * \texttt{fact(2)} \\
&= 3 * (2 * \texttt{fact(1)}) \\
&= 3 * (2 * (1 * \texttt{fact(0)})) \\
&= 3 * (2 * (1 * 1)) \\
&= 3 * (2 * 1) \\
&= 3 * 2 \\
&= 6
\end{aligned}$$

3.2 Example: Fibonacci

$$\begin{aligned}
\texttt{fib(3)} &= \texttt{fib(2)} + \texttt{fib(1)} \\
&= \big(\texttt{fib(1)} + \texttt{fib(0)}\big) + \texttt{fib(1)} \\
&= \big(1 + \texttt{fib(0)}\big) + \texttt{fib(1)} \\
&= (1 + 0) + \texttt{fib(1)} \\
&= 1 + \texttt{fib(1)} \\
&= 1 + 1 \\
&= 2
\end{aligned}$$