Fruitful Functions

Up to this point functions have been “commands”

* Without the turtle drawing, how would we know if our function did something?
* We can have our function yield a result we can check

Consider the following

* Turtle.forward(100) #moves the turtle
* Math.sqrt(2) #does the squareroot of 2

To see what a fruitful function does we must see what it yields

* Also known as what it returns
* Using the keyword return you can yield a value from a function

Example:

* Def addOne(x):
  + Return x + 1
* When I call addOne(2) it will return 3
  + Basically the function call will get replaced with the return value!

Classic Fruitful Function Examples

Factorial

Def fact(n):

If n==0:

Return 1

Else:

Return\*(fact(n-1))

Fibonacci sequence

Def fib(n):

If n==0:

Return 0

Elif n==1:

Return 1

Else:

Return fib(n-1)+fib(n-2)

Substitution Trace

Execution traces are good, but:

* Can be long
* Can contain more detail than we need

Substitution trace can be simpler!

* Substitution tracing is a generalization of arithmetic expressions evaluation
* Very similar to how you solve a math expression!

Example:

Fact(3) =3 \* fact(2)

=3\*(2\*fact(1))

=3\*(2\*(1\*fact(0))

=3\*(2\*(1\*1))

=3\*(2\*1)

=3\*2

=6

Fib (3) =fib(2)+fib(1)

=(fib(1)+fib(0))+fib(1)

=(1+fib(0))+fib(1)

=(1+0)+fib(1)

=1+fib(1)

=1+1

=2

Tail-Recursive Fruitful Functions and Iteration

Example:

Def factAccum(n,a)

“””factAccum: NatNum \* NatNum -> NatNum”””

If n==0:

Return a

Else:

Return factAccum(n-1,n\*a)

Def fact(n):

“””fact: NatNum -> NatNum”””

Return factAccum(n,1)

Def fibAccum(n, a, b):

“””fibAccum: NatNum \* NatNum \* Natnum -> NatNum”””

If n==0:

Return a

Elif n==1:

Return b

Else:

Return fibAccum(n-1,b,a+b)

Def fib(n):

“””fib: NatNum -> NatNum”””

Return fibAccum(n,0,1)

Iterative Formulation

* Since tail-recursion can be iteration
* Can we make these also iteratively?
  + Note we do not need break since return exits the function

factAccum(n,a):

“””factAccum: NatNum \* NatNum -> NatNum”””

While true:

If n==0:

Return a

Else:

A=n\*a

N=n-1

Def fact(n):

“””fact: NatNum -> NatNum”””

Return factAccum(n,1)

Iterative Formulation with while condition and assignment

Def fact(n):

“””fact: NatNum -> NatNum”””

A=1

While n>0:

A=n\*a

N=n-1

Return a

Def fib(n):

A=0  
 B=1

While n>1:

N=n-1

Newa=b

Newb=a+b

Timelines

Timelines can be used instead of substitution traces

Substitution traces are not really good for iteration tracing

Timelines trace variable changes from one iteration to the next

Timeline example for fib(5)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N | - | 4 |  |  |  |  |
| Newa | - | 1 |  |  |  |  |
| Newb | - | 1 |  |  |  |  |
| A | 0 | 1 |  |  |  |  |
| B | 1 | 1 |  |  |  |  |