A *very* simple function

In [ ]:

```
incr(x) = x + 1
```

In [ ]:

```
incr(1)
```

Lets look at the code generated

In [ ]:

```
@code_typed incr(1)
```

In [ ]:

```
@code_llvm incr(1)
```

In [ ]:

```
@code_native incr(1)
```

Call with a real number

In [ ]:

```
incr(2.3)
```

In [ ]:

```
@code_typed incr(2.3)
```

In [ ]:

```
@code_llvm incr(2.3)
```

In [ ]:

```
@code_native incr(2.3)
```

Derived types

Complex number

In [ ]:

```
incr(2 + 3im)
```

In [ ]:

```
@code_native incr(2 + 3im)
```

In [ ]:

```
@code_native incr(2 + 3.0im)
```

Rationals

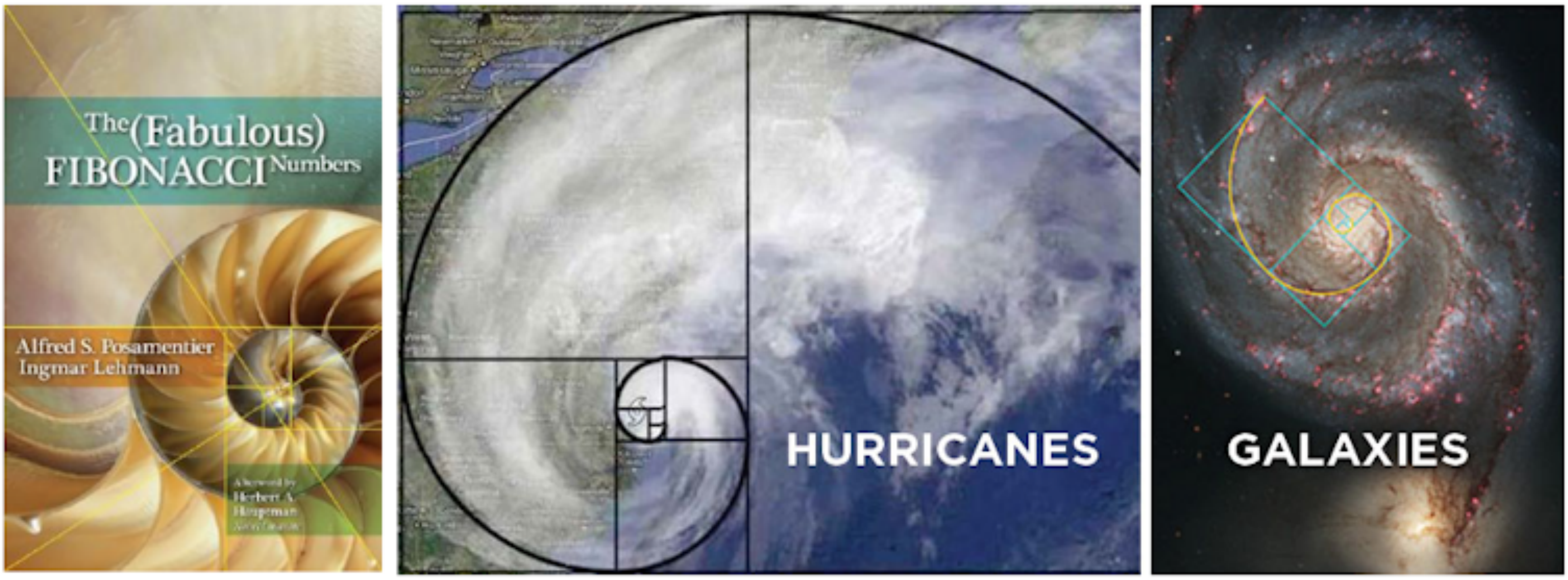
In [ ]:

```
incr(2//3)
```

In [ ]:

```
@code_native incr(2//3)
```

Fibonacci Sequences



The Fibonacci Sequence is the series of numbers:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

$F_1 = F_2 = 1$

$F_n = F_{n-1} + F_{n-2}$ for $n > 2$

First attempt

In [ ]:

```
function fib1(n::Integer)
    @assert n > 0
    (n < 3) ? 1 : fib1(n-1) + fib1(n-2)
end
```

In [ ]:

```
fib1(43)
```

In [ ]:

```
@code_native fib1(402)
```

A better version

In [ ]:

```
# Need BIG arithmetic to avoid overflows

function fib2(n::Integer)
  @assert n > 0
  (a, b) = (big(0), big(1))
  while n > 0
    (a, b) = (b, a+b)
    n -= 1
  end
  return a
end
```

In [ ]:

```
fib2(402)
```

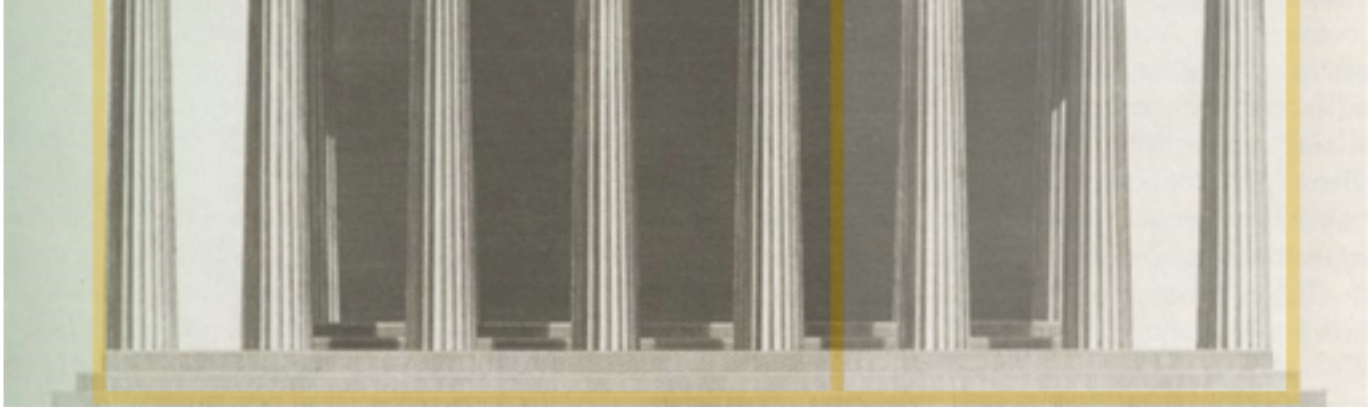
In [ ]:

```
@code_native fib2(402)
```

In [ ]:

```
@elapsed fib2(402)
```

Golden Ratio



In [ ]:

```
#
# Golden ratio is 1.61803398875 <=> (1 + √5)/2
#
ψ = fib2(100)/fib2(99)
```

Computing a Julia Set

In [ ]:

```
function juliaset(z, z0, nmax::Int64)
    for n = 1:nmax
        if abs(z) > 2 (return n-1) end
        z = z^2 + z0
    end
    return nmax
end
```

In [ ]:

```
@code_native juliaset(0.0+0.0im, -0.8+0.16im, 256)
```

In [ ]:

```
include("./pgmfile.jl")
```

In [ ]:

```
h = 400; w = 800;
m = Array(Int64, h, w);
c0 = -0.8+0.16im;
pgm_name = "juliaset.pgm";

tic();
for y=1:h, x=1:w
    c = complex((x-w/2)/(w/2), (y-h/2)/(w/2))
    m[y,x] = juliaset(c, c0, 256)
end
toc();
create_pgmfile(m, pgm_name);
```

In [ ]:

```
; display ./juliaset.pgm
```

Asian Option

In [ ]:

```
using Winston, Colors
```

In [ ]:

```
S0  = 100;      # Spot price
K   = 102;      # Strike price
r   = 0.05;     # Risk free rate
q   = 0.0;      # Dividend yield
v   = 0.2;      # Volatility
tma = 0.25;     # Time to maturity

T   = 90;       # Number of time steps
dt  = tma/T;    # Time increment

x = linspace(1,T);
```

Five random walks

In [ ]:

```
for k = 1:5
  S = zeros(Float64,T)
  S[1] = S0;
  dW = randn(T)*sqrt(dt);
  [ S[t] = S[t-1] * (1 + (r - q - 0.5*v*v)*dt + v*dW[t] + 0.5*v*v*dW[t]*dW[t]) fo
  p = FramedPlot(title = "Simulation of Asian Options")
  add(p, Curve(x,S,color=parse(Colorant,"red")))
  display(p)
  readline()
end
```

In [ ]:

```
function asianOpt(N::Integer, T::Integer; S0 = 100.0, K=100.0, r=0.25, q=0.0, v=0

# Initialize the terminal stock price matrices

  @assert N > 0;
  @assert T > 0;

  S = zeros(Float64,N,T);
  dt = tma/T;

  for n=1:N
    S[n,1] = S0;
  end

# Simulate the stock price and compute...
# ... the average of terminal stock price.

  A = zeros(Float64,N);

  for n=1:N
    dW = randn(T)*sqrt(dt);
    for t=2:T
      z0 = (r - q - 0.5*v*v)*S[n,t-1]*dt;
      z1 = v*S[n,t-1]*dW[t];
      z2 = 0.5*v*v*S[n,t-1]*dW[t]*dW[t];
      S[n,t] = S[n,t-1] + z0 + z1 + z2;
    end
    A[n] = mean(S[n,:]);
  end

# Define the payoff

  P = zeros(Float64,N);

  for n = 1:N
    P[n] = max(A[n] - K, 0);
  end

# Calculate the price of the option.

  price = exp(-r*tma)*mean(P);
  return price
end
```

In [ ]:

```
@elapsed asianOpt(100000,100,K=102)
```

In [ ]:

```

```