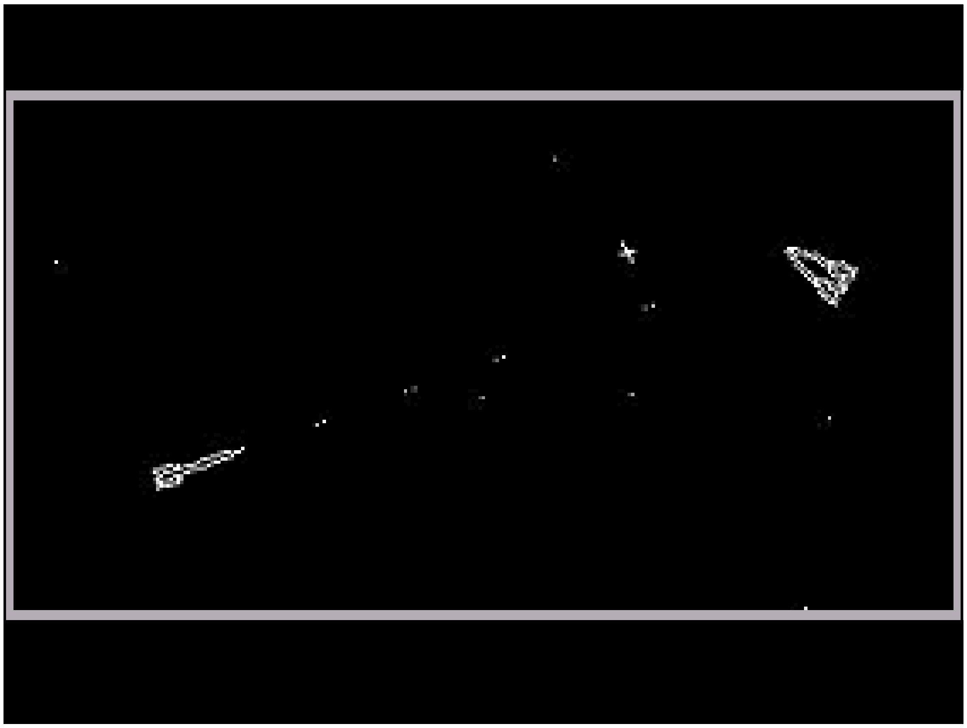
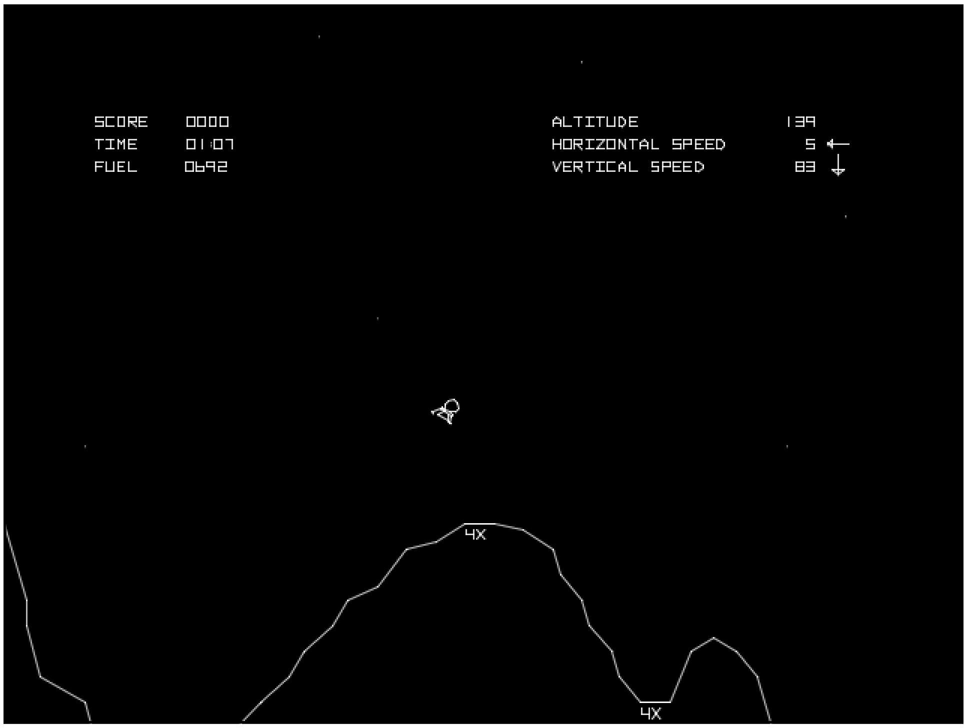
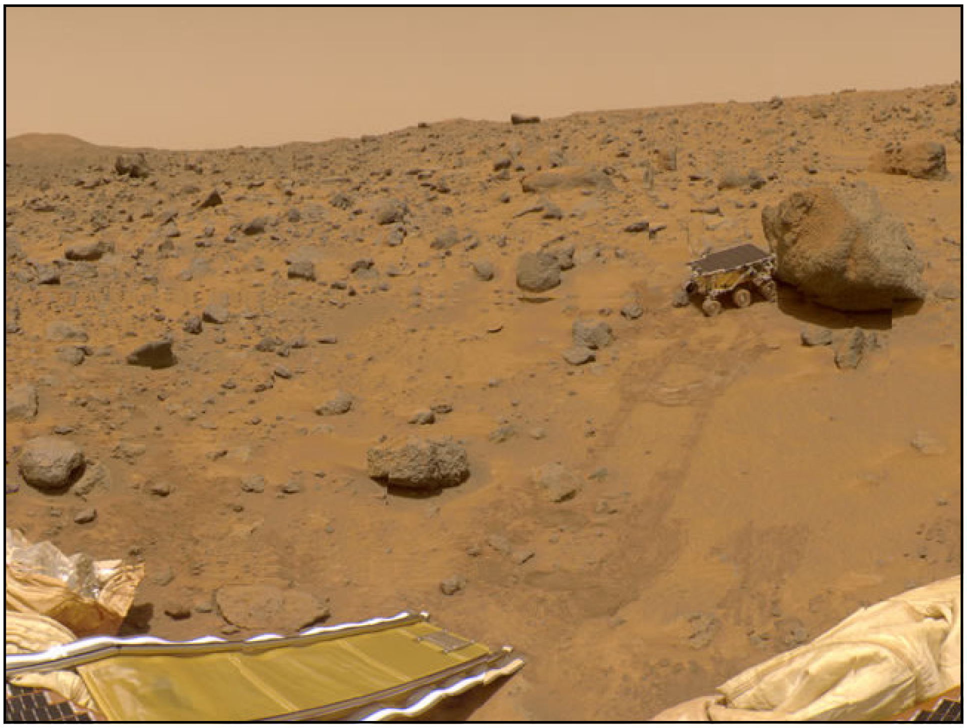
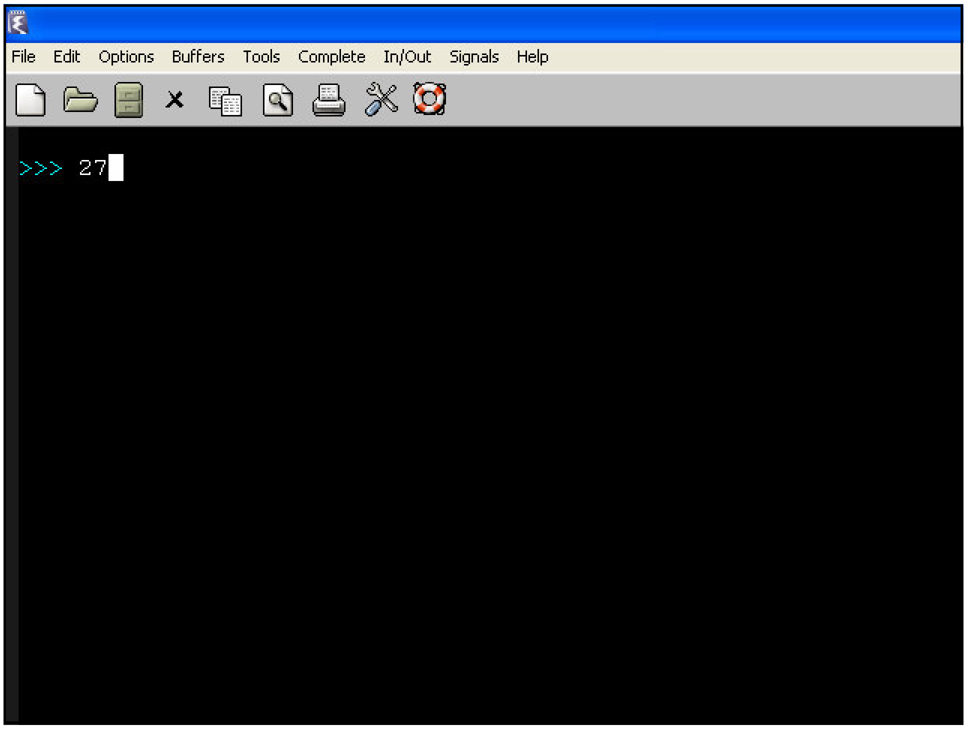


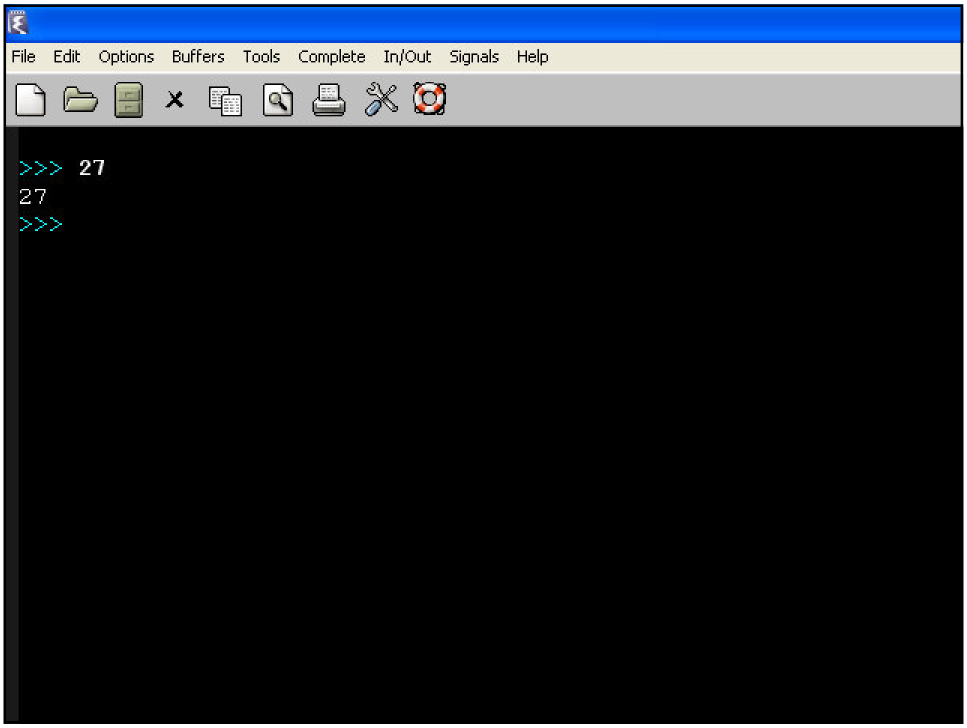

SCORE 0000
TIME 01:07
FUEL 0692
ALTITUDE 139
HORIZONTAL SPEED 5 ←
VERTICAL SPEED 83 ↓
4X
4X


STAR WARS
EMPIRE AT WAR
SINGLE PLAYER
NEW CAMPAIGN
GALACTIC CONQUEST
SKIRMISH BATTLES
BACK
LUCASARTS

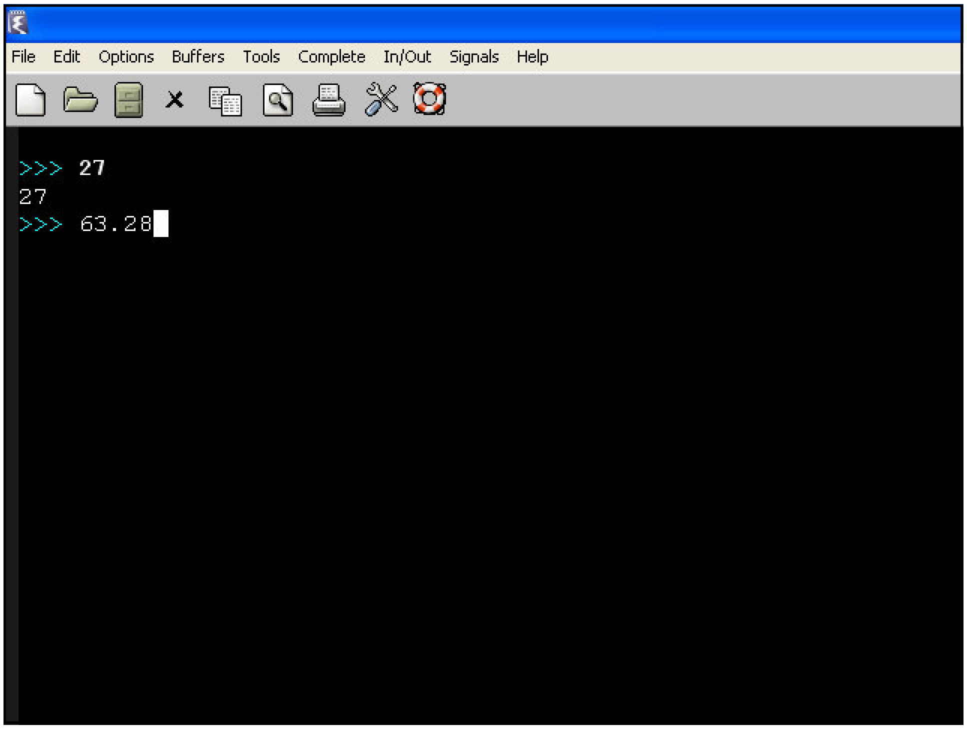



File
Edit
Options
Buffers
Tools
Complete
In/Out
Signals
Help

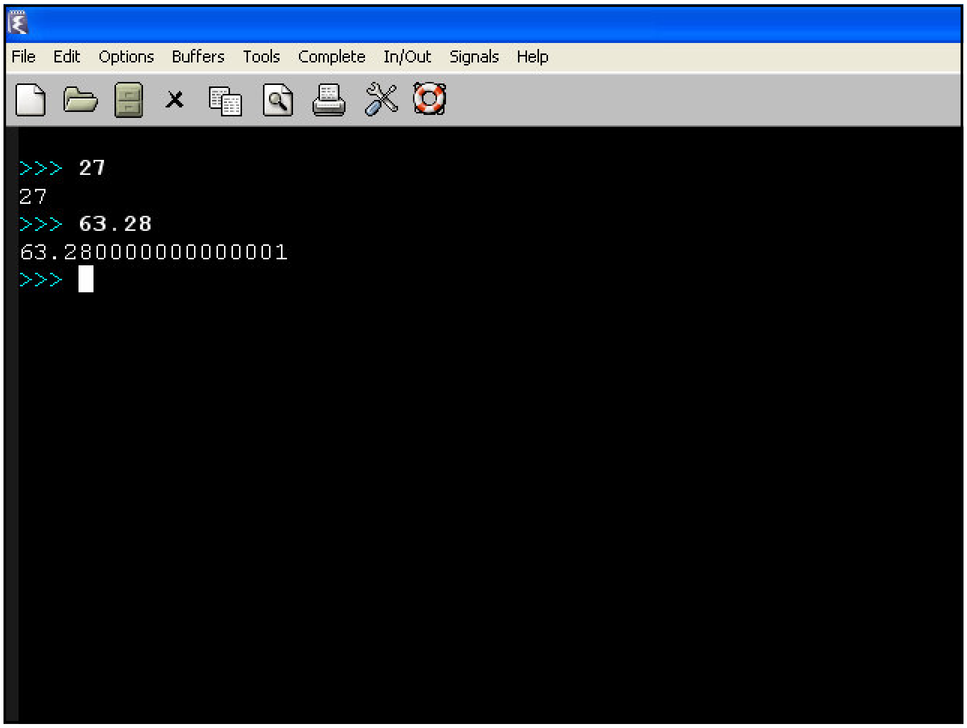
```
>>> 27
```


File
Edit
Options
Buffers
Tools
Complete
In/Out
Signals
Help

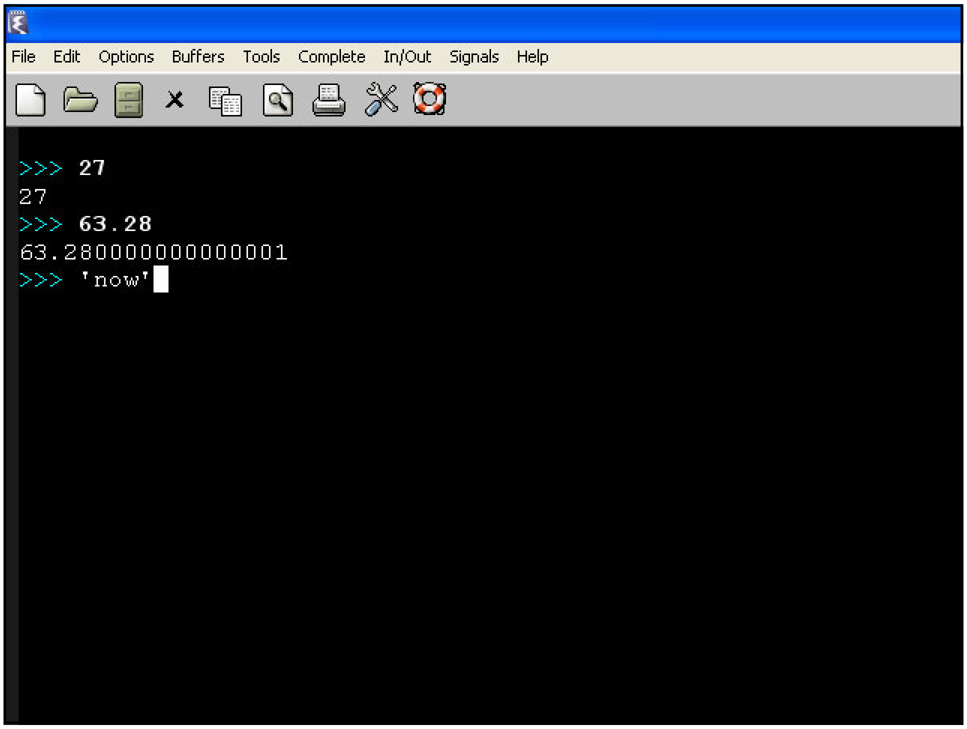
```
>>> 27
27
>>>
```



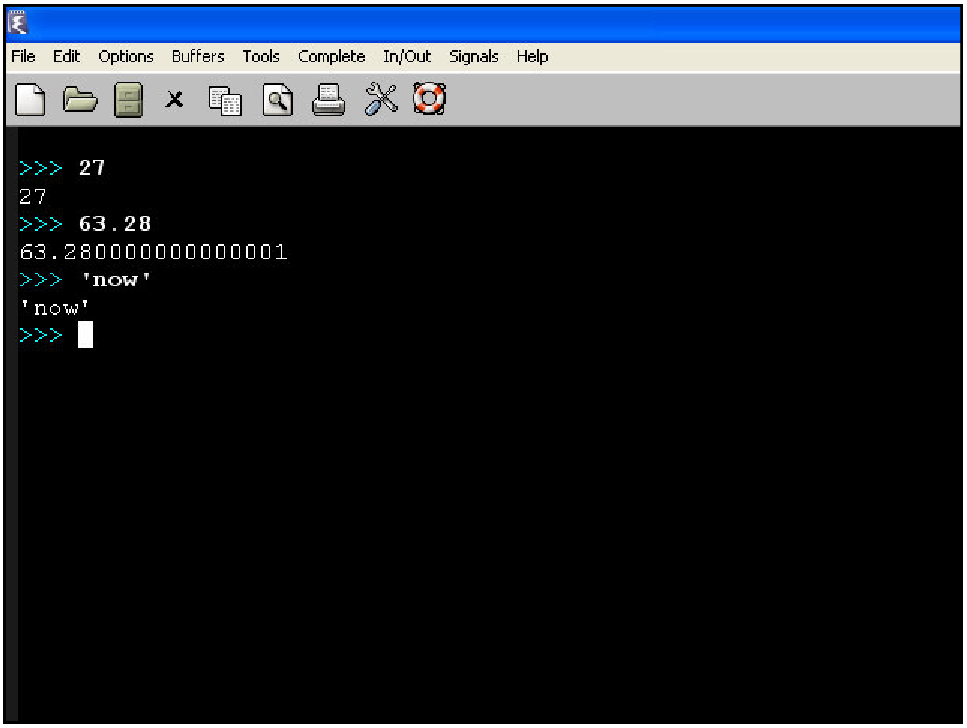
```
>>> 27
27
>>> 63.28
```



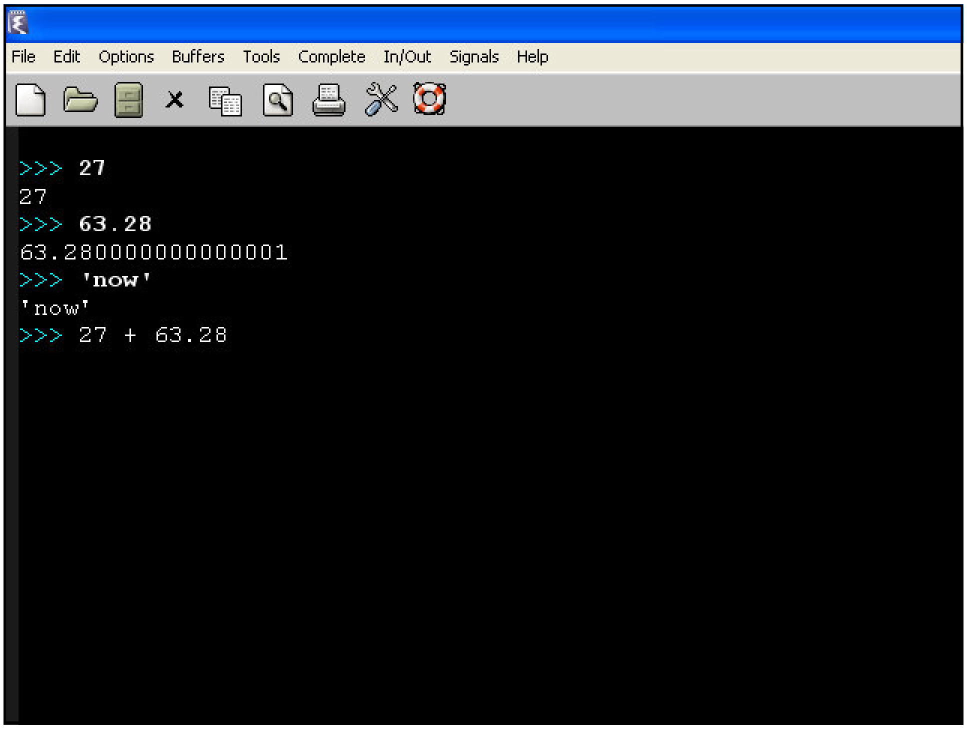
```
>>> 27
27
>>> 63.28
63.280000000000001
>>> 
```



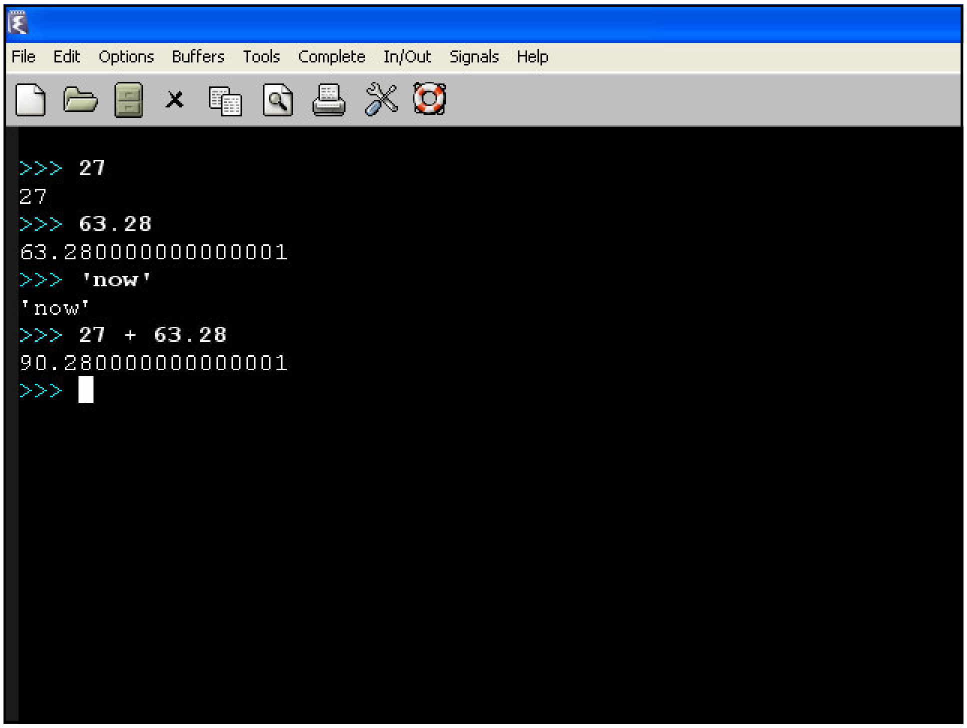
```
>>> 27
27
>>> 63.28
63.280000000000001
>>> 'now'
```



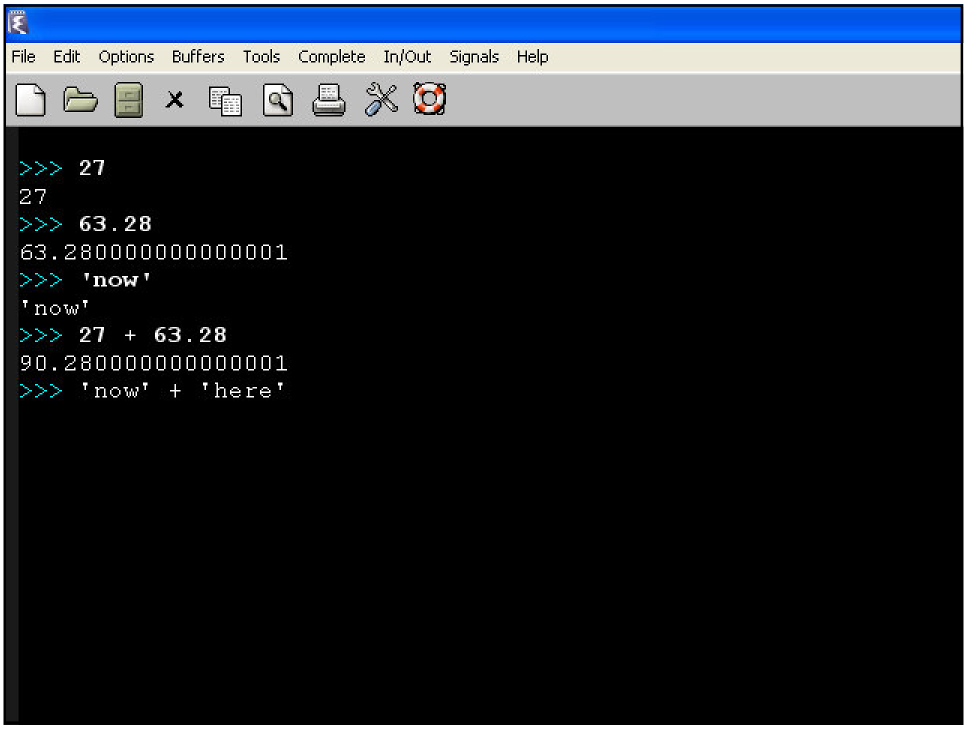
```
>>> 27
27
>>> 63.28
63.280000000000001
>>> 'now'
'now'
>>> 
```



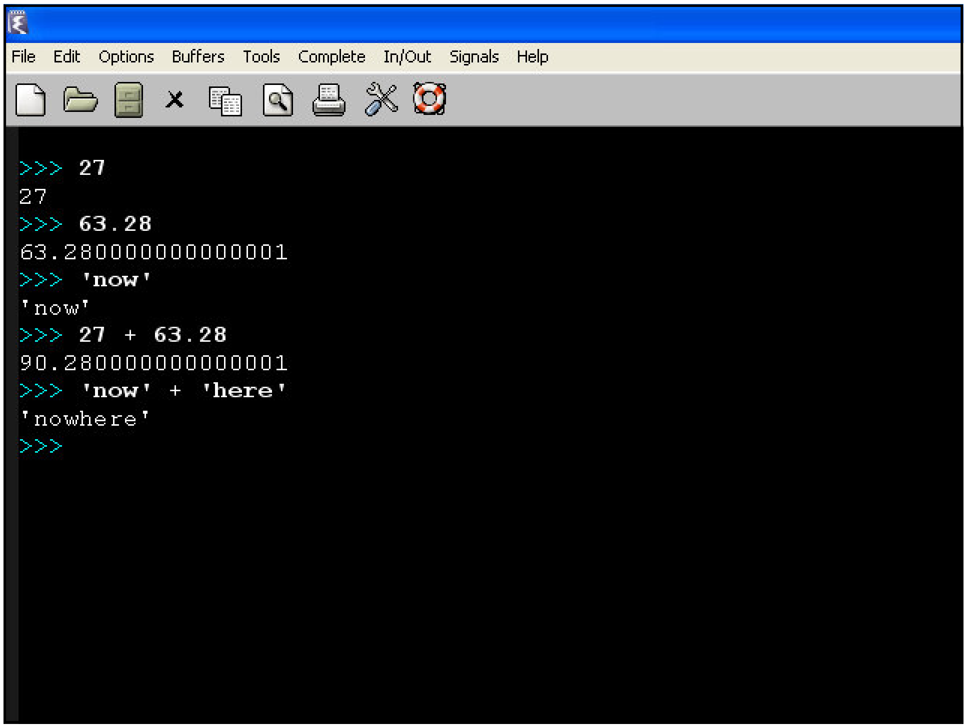
```
>>> 27
27
>>> 63.28
63.280000000000001
>>> 'now'
'now'
>>> 27 + 63.28
```



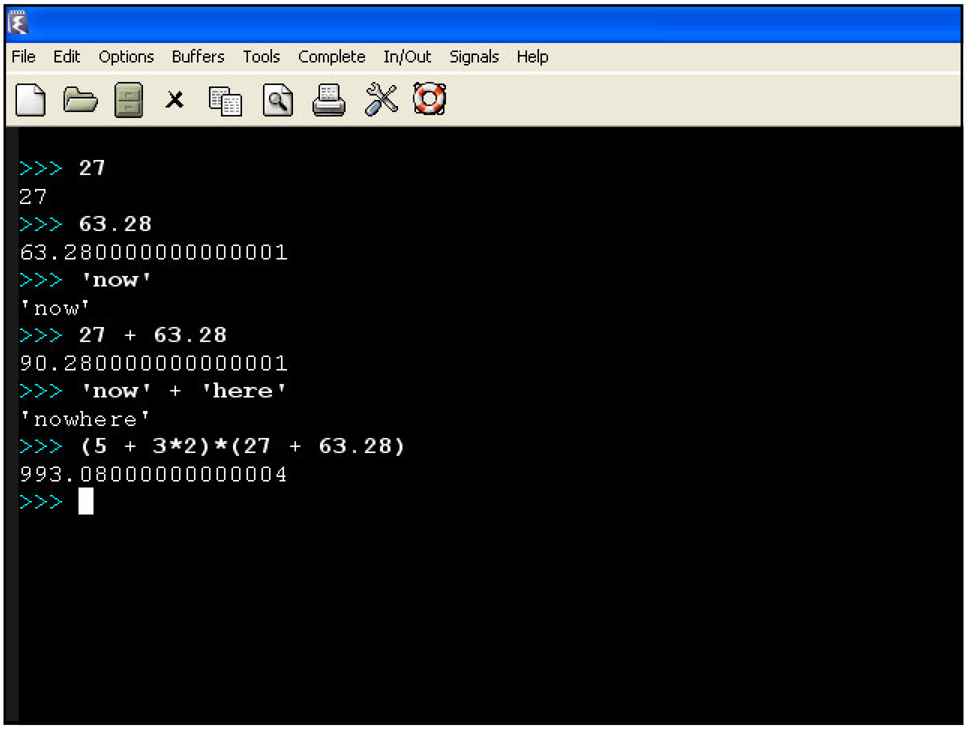
```
>>> 27
27
>>> 63.28
63.280000000000001
>>> 'now'
'now'
>>> 27 + 63.28
90.280000000000001
>>> 
```



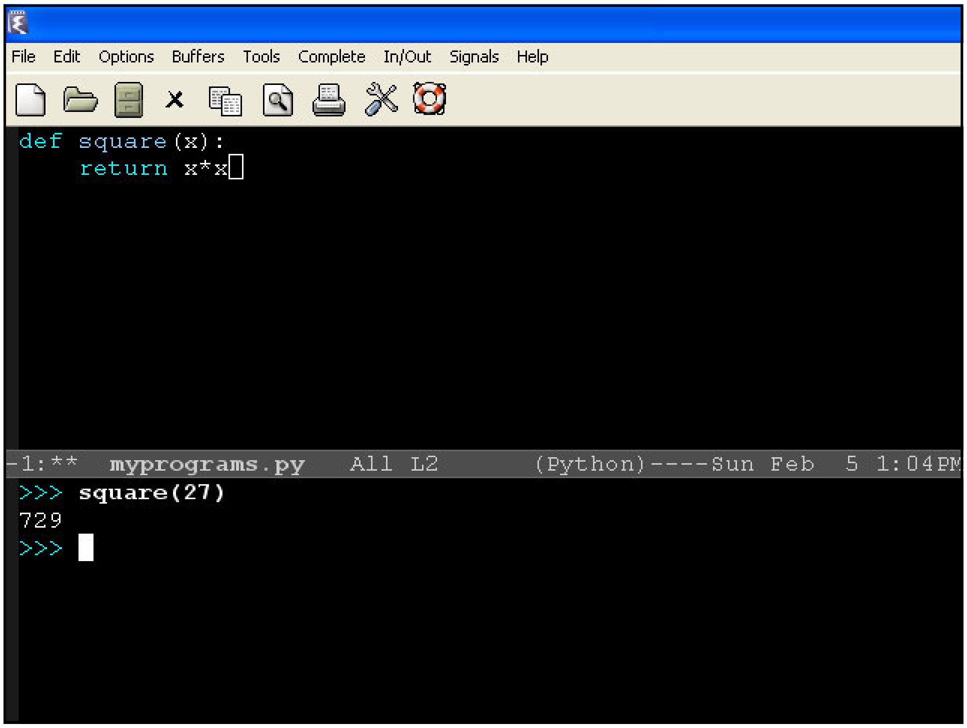
```
>>> 27
27
>>> 63.28
63.280000000000001
>>> 'now'
'now'
>>> 27 + 63.28
90.280000000000001
>>> 'now' + 'here'
```



```
>>> 27
27
>>> 63.28
63.280000000000001
>>> 'now'
'now'
>>> 27 + 63.28
90.280000000000001
>>> 'now' + 'here'
'nowhere'
>>>
```



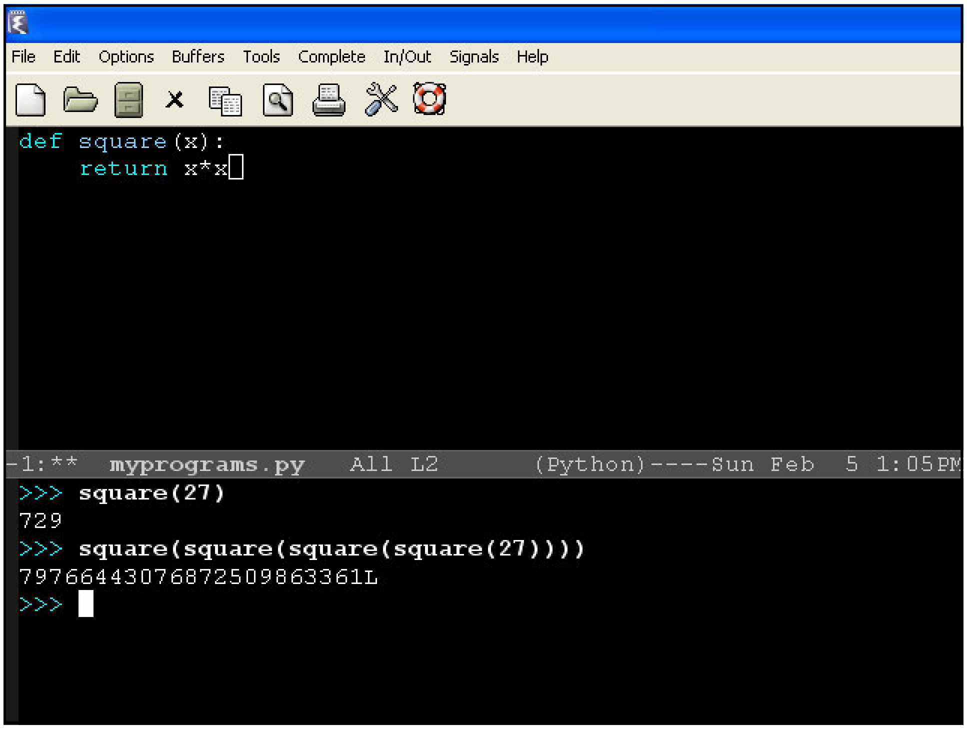
```
>>> 27
27
>>> 63.28
63.280000000000001
>>> 'now'
'now'
>>> 27 + 63.28
90.280000000000001
>>> 'now' + 'here'
'nowhere'
>>> (5 + 3*2)*(27 + 63.28)
993.08000000000004
>>> 
```



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

```
-1:**  myprograms.py    All L2        (Python)----Sun Feb  5  1:04PM
```

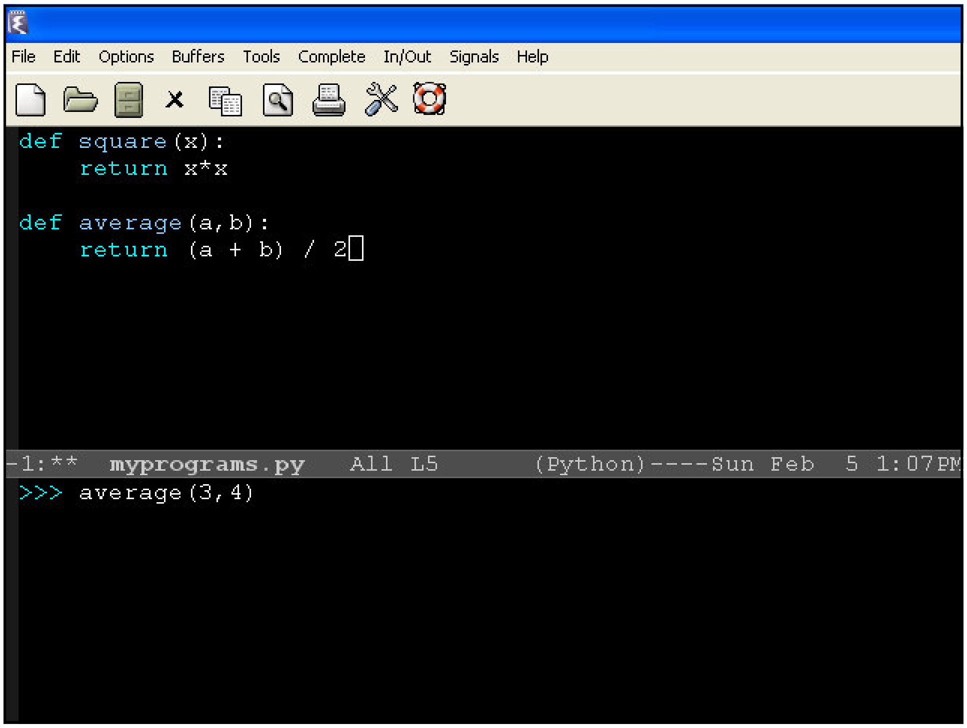
```
>>> square(27)
729
>>> 
```



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

```
-1:**  myprograms.py    All L2       (Python)----Sun Feb  5 1:05PM
```

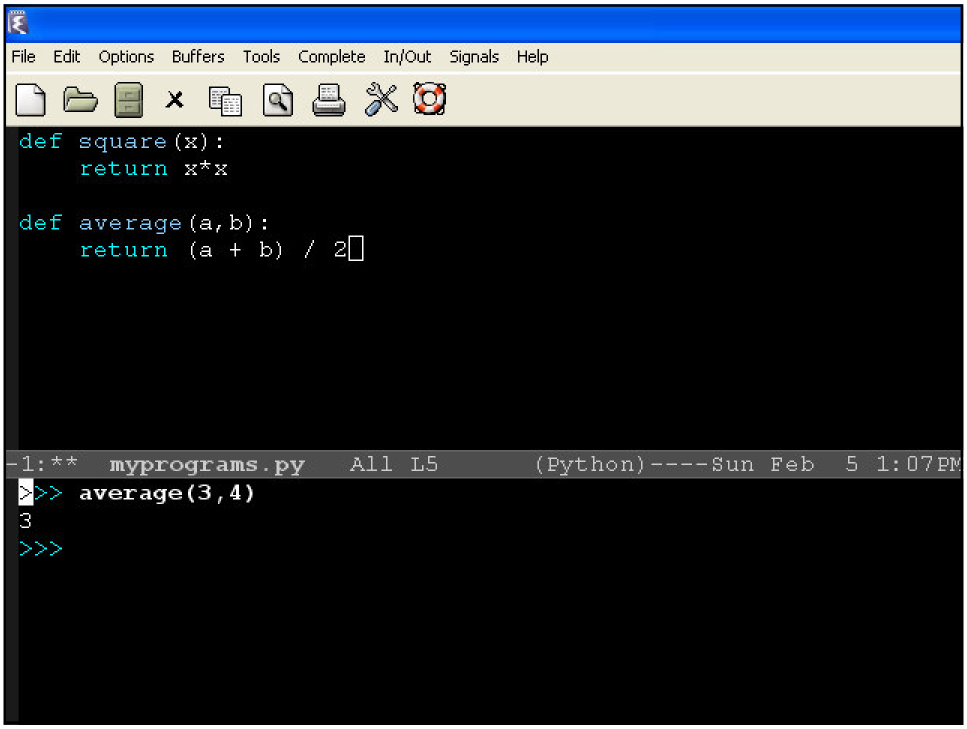
```
>>> square(27)
729
>>> square(square(square(square(27))))
79766443076872509863361L
>>> 
```



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

def average(a,b):
    return (a + b) / 2
```

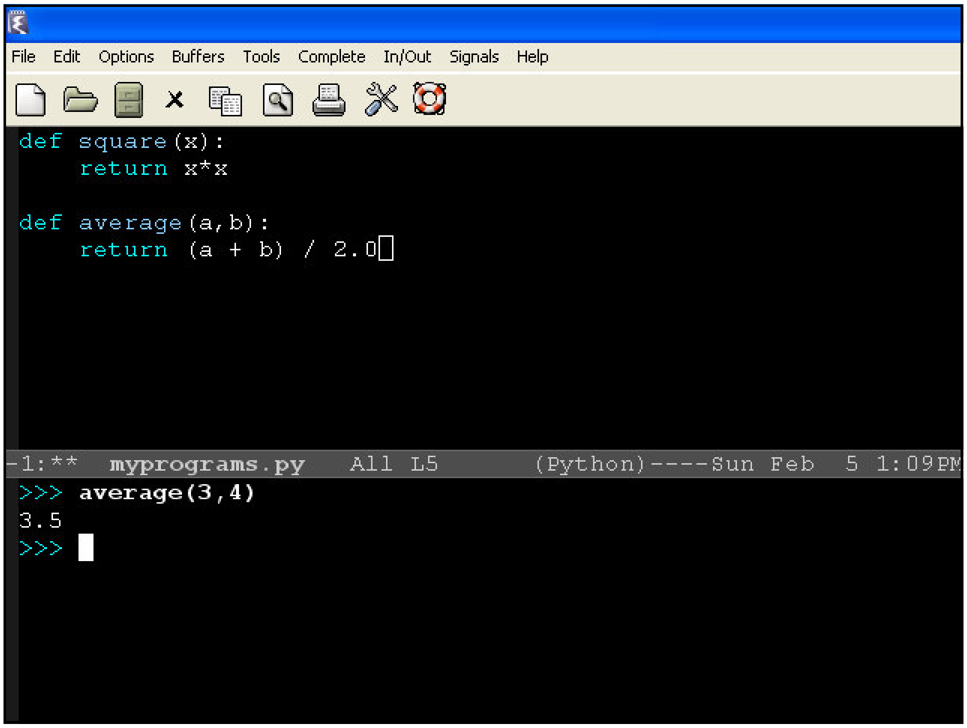
```
-1:**  myprograms.py    All L5       (Python)----Sun Feb  5 1:07PM
```

```
>>> average(3,4)
```



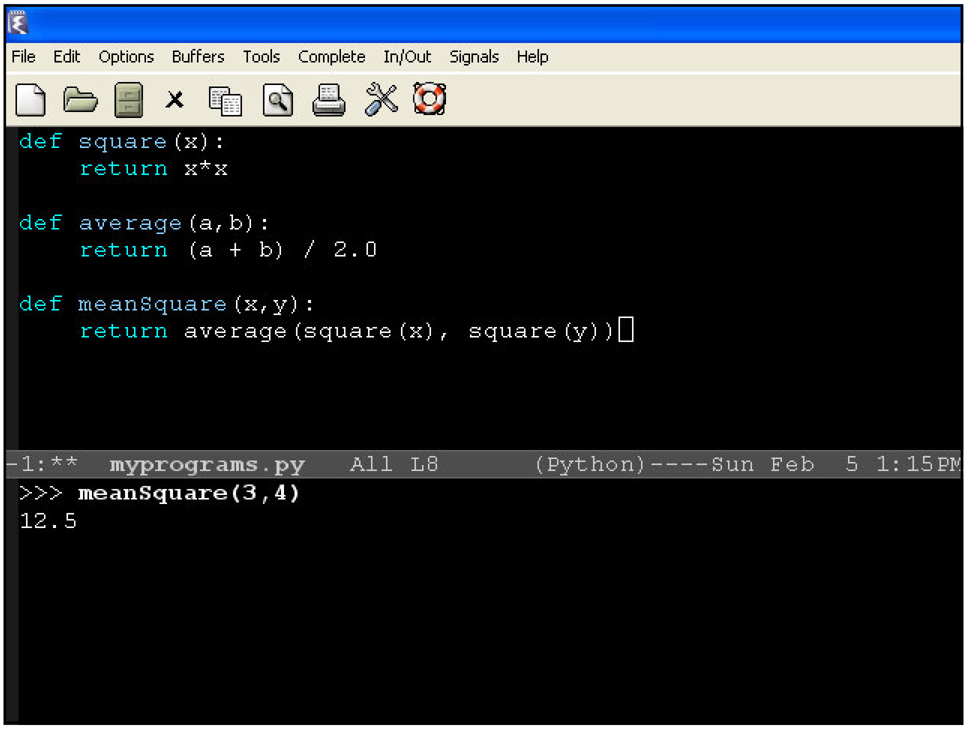
```
def square(x):
    return x*x

def average(a,b):
    return (a + b) / 2
```

```
-1:**  myprograms.py   All L5      (Python)----Sun Feb  5 1:07PM
>>> average(3,4)
3
>>>
```



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

def average(a,b):
    return (a + b) / 2.0
```

```
-1:**  myprograms.py   All L5      (Python)----Sun Feb  5 1:09PM
>>> average(3,4)
3.5
>>>
```



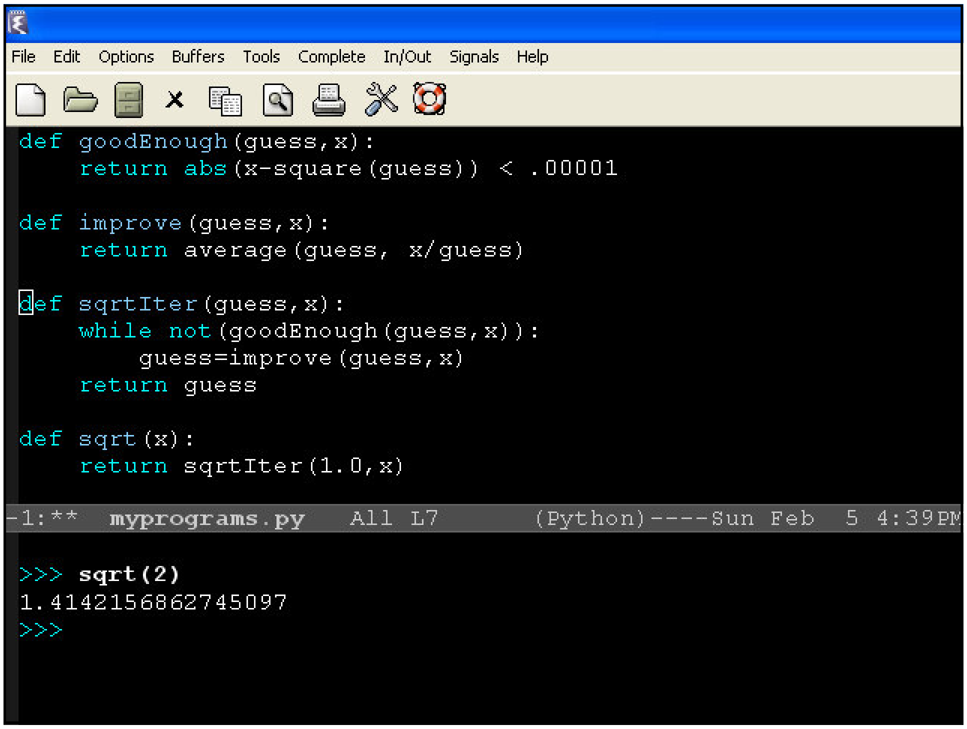
```
def square(x):
    return x*x

def average(a,b):
    return (a + b) / 2.0

def meanSquare(x,y):
    return average(square(x), square(y))
```

```
-1:**  myprograms.py    All L8        (Python)----Sun Feb  5 1:15PM
```

```
>>> meanSquare(3,4)
12.5
```

Computing square roots

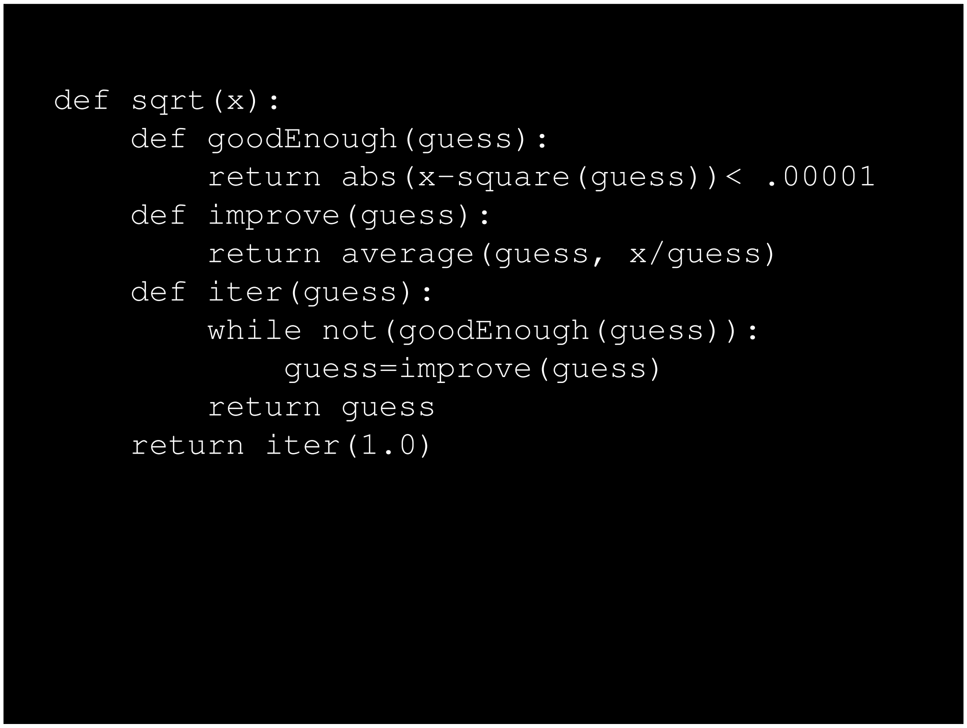
- To compute an approximation to the square root of x:
  - Let g be a guess for the answer
  - Compute an improved guess by taking the average of g and x/g
  - Keep improving the guess until it’s good enough.



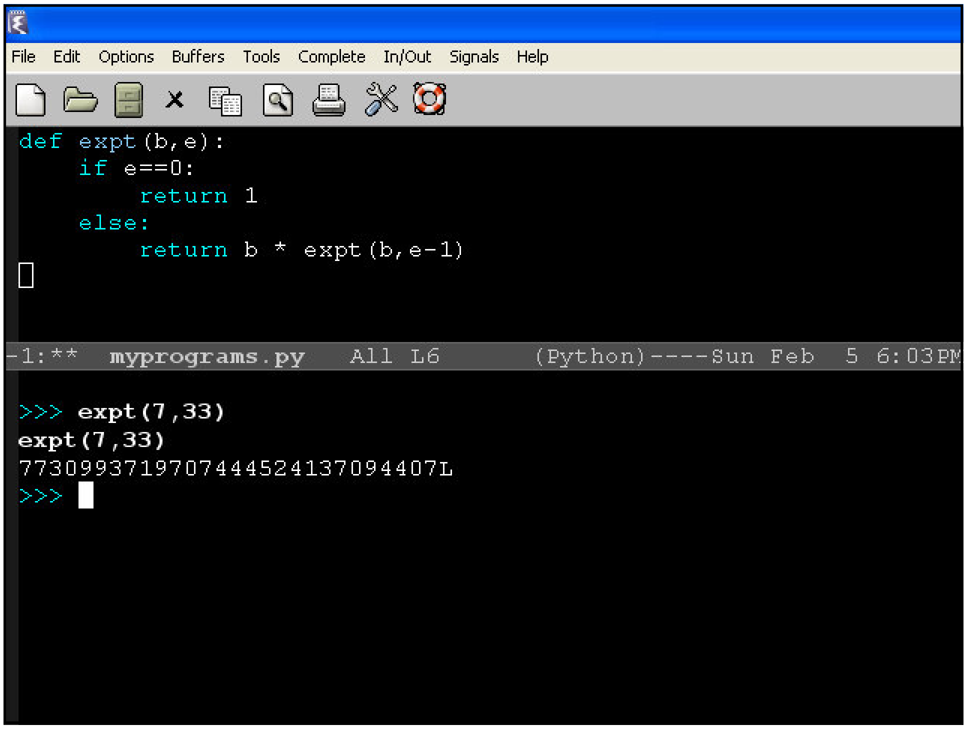
```
def goodEnough(guess,x):
    return abs(x-square(guess)) < .00001

def improve(guess,x):
    return average(guess, x/guess)

def sqrtIter(guess,x):
    while not(goodEnough(guess,x)):
        guess=improve(guess,x)
    return guess

def sqrt(x):
    return sqrtIter(1.0,x)
```

```
-1:**  myprograms.py    All L7      (Python)----Sun Feb  5 4:39PM
```

```
>>> sqrt(2)
1.4142156862745097
>>>
```



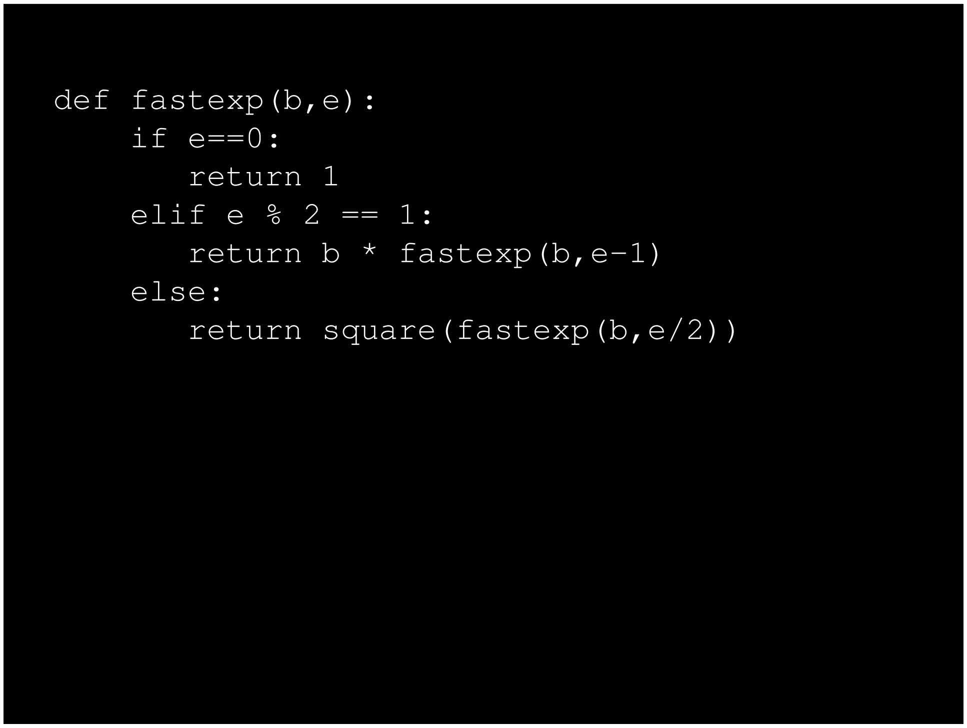
```
def sqrt(x):
    def goodEnough(guess):
        return abs(x-square(guess))< .00001
    def improve(guess):
        return average(guess, x/guess)
    def iter(guess):
        while not(goodEnough(guess)):
            guess=improve(guess)
        return guess
    return iter(1.0)
```



```
def expt(b,e):
    if e==0:
        return 1
    else:
        return b * expt(b,e-1)
```

```
-1:**  myprograms.py    All L6       (Python)----Sun Feb  5 6:03PM
```

```
>>> expt(7,33)
expt(7,33)
7730993719707444524137094407L
>>>
```



```
def fastexp(b,e):
    if e==0:
       return 1
    elif e % 2 == 1:
       return b * fastexp(b,e-1)
    else:
       return square(fastexp(b,e/2))
```

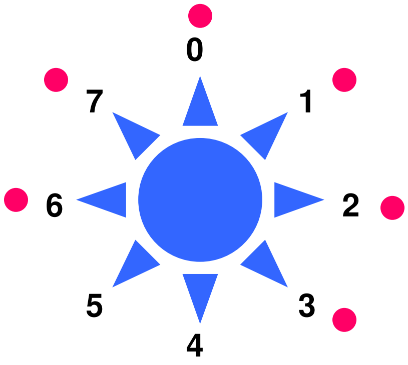
Order of growth

For a process that uses resources $R(n)$ for a problem of size n, we say that $R(n)$ has *order of growth* $\Theta(f(n))$ if there are positive constants k_1 and k_2 independent of n such that

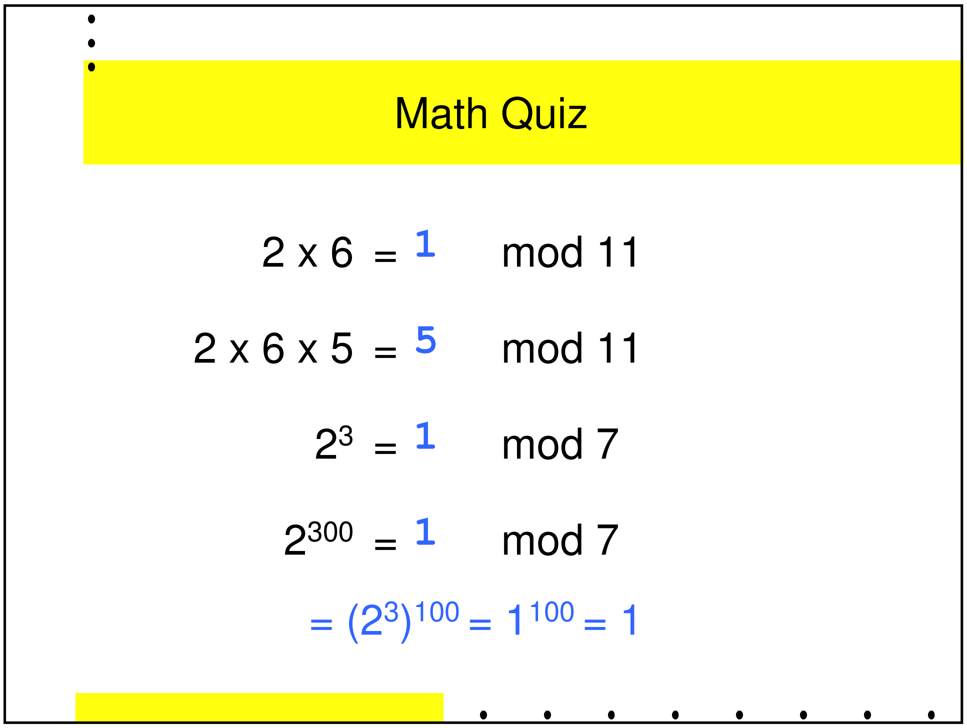
$$k_1 f(n) \leq R(n) \leq k_2 f(n)$$

for *n* sufficiently large

Modular arithmetic



6 + 5 = 3 (mod 8)



Math Quiz

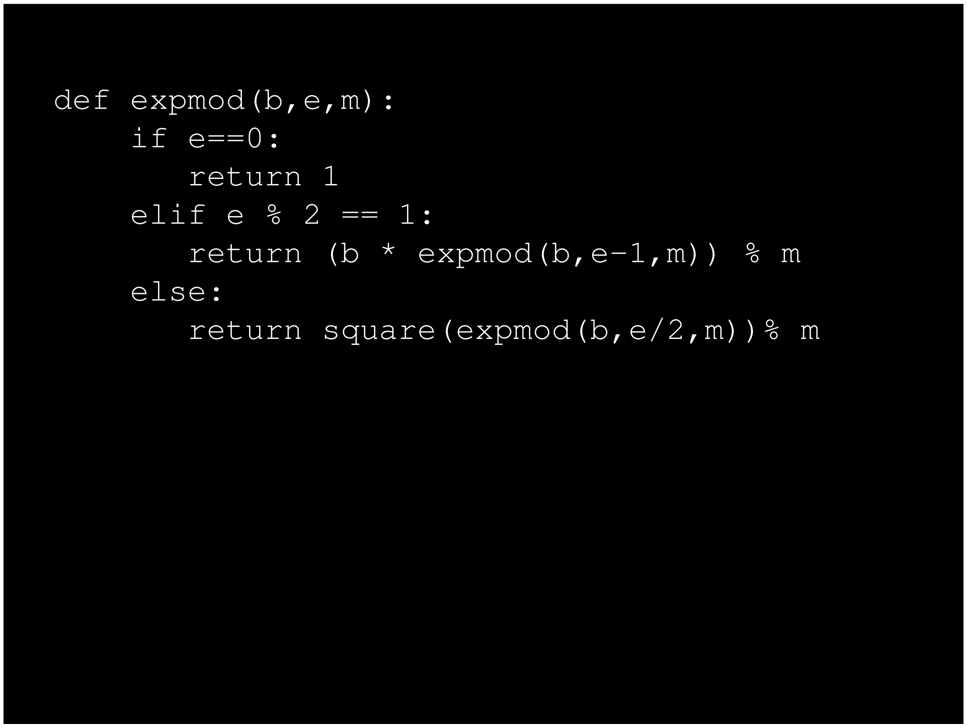
$2 \times 6 = 1 \mod 11$

$2 \times 6 \times 5 = 5 \mod 11$

$2^3 = 1 \mod 7$

$2^{300} = 1 \mod 7$

$= (2^3)^{100} = 1^{100} = 1$



```
def expmod(b,e,m):
    if e==0:
       return 1
    elif e % 2 == 1:
       return (b * expmod(b,e-1,m)) % m
    else:
       return square(expmod(b,e/2,m))% m
```

Expmod lets you compute powers modulo m in order log n steps

- Problem: Given *a* and *p* and *x*, find *y* such that

$$a^x = y \pmod{p}$$

- Use expmod
- Example: If *x* is a 500-digit number, we can compute $a^x \pmod{p}$ in about 1700 ($= \log_2 10^{500}$) steps.

There’s no shortcut for computing logarithms mod *p*

- Problem: Given *a* and *p* and *y*, find *x* such that

$$a^x = y \pmod{p}$$

- As far as anyone knows, there are no shortcuts.
  - The only way to do this is essentially by brute-force search among all possibilities for *x.*
- Example: If *p* is a 500-digit number, finding *x* so that

$$a^x = y \pmod{p}$$

requires about 10^{500} steps.

Secret communication on public channels

- Alice and Bob want to create a shared secret number that they can use as a cryptographic key. But *all* of their communications can be overhead.
- There is a great idea
- Alice and Bob can create a shared secret key, even if they have never met before and have made no prior arrangements, and even if everyone can eavesdrop on ***all*** their communications …
- … including eavesdropping on the communications Alice and Bob use to establish the key!

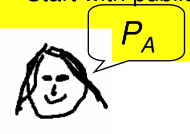
Public-Key Cryptography

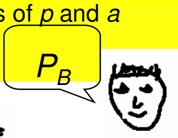
- Whit Diffie and Marty Hellman, *New Directions in Cryptography,* 1976
- Clifford Cocks and Malcolm Williamson, secret work in the British GCHQ, 1973-74, revealed only in 1997

Diffie-Hellman Key Agreement

Start with public, standard values of *p* and *a*

Alice





Bob

Pick a secret number S_A

Compute $P_A = a^{S_A} \bmod p$

Shout out P_A

Compute $P_B^{S_A} \bmod p$

Eve

Pick a secret number S_B

Compute $P_B = a^{S_B} \bmod p$

Shout out P_B

Compute $P_A^{S_B} \bmod p$

Main point: Alice and Bob have computed the same number, because

$$\left(P_B^{S_A} = (a^{S_B})^{S_A} = a^{S_B S_A} = (a^{S_A})^{S_B} = P_A^{S_B}\right) \bmod p$$

Alice and Bob can now use this number as a shared key for encrypted communication

Eavesdroppers know $P_A = a^{S_A} \bmod p$ and $P_B = a^{S_B} \bmod p$

But going from these to $a^{S_A S_B} \bmod p$ requires computing logarithms mod *p*, as far as anyone knows

Basic result of Diffie-Hellman

- As a consequence, someone eavesdropping on Alice and Bob would require exponentially more computer power to break the system, than Alice and Bob require to establish a key.

	Procedures	Data
Primitives	+, *, ==, …	numbers, strings
Means of combination	if, while, … composition, e.g., can write 3*(4+7)	lists
Means of abstraction	def	??
Capturing common patterns	???	??

END