Recursion

It is a programming technique where a function calls itself as part of the program

```
def func(...):
    :
    :
    func(...)
```

recursion

In order to stop the recursion you need to add an "end condition" that will prevent the function to call itself when the solution is obtained.

Solving factorial(n) using recursion

73

$$fact(n) = \begin{cases} n \times fact(n-1) & \text{if } n>1 \\ 1 & \text{if } n==1 \end{cases}$$

$fact(5) = 5 \times 4 \times 3 \times 2 \times 1$

$= 5 \times \underbrace{fact(4)}$

$= 5 \times \underbrace{4 \times fact(3)}$

$5 \times 4 \times 3 \times fact(2)$

$5 \times 4 \times 3 \times 2 \times fact(1)$

```
def fact(n):
    #end condition
    if n==1:
      return 1
    else
      return n*fact(n-1)


def main():
    n=eval(input("n?"))
    print("fact(", n, ")=", fact(n))
main()
```

Divide and Conquer Strategy

Given a complex problem

- Divide problem in smaller problems
- Call the function recursively with smaller problems.
- Put the smaller solutions together to form the bigger solution.

fact(5)

↓

5 * fact(4)

5 * 4 * fact(3)

5 * 4 * 3 * fact(2)

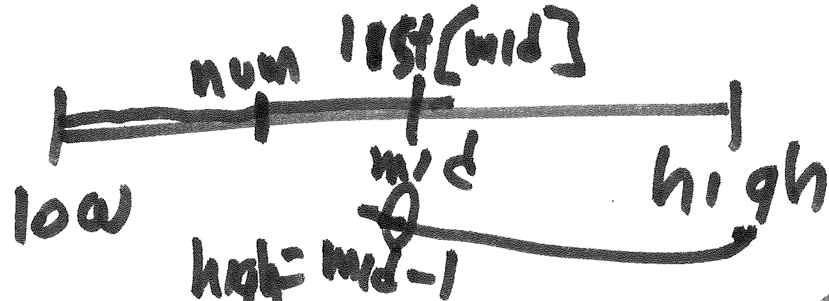
5 * 4 * 3 * 2 * fact(1)

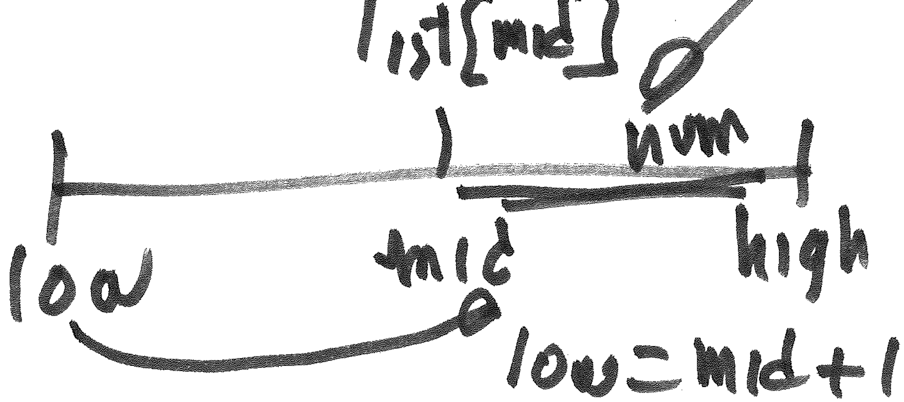
5 * 4 * 3 * 2 * 1

Making binary Search recursive

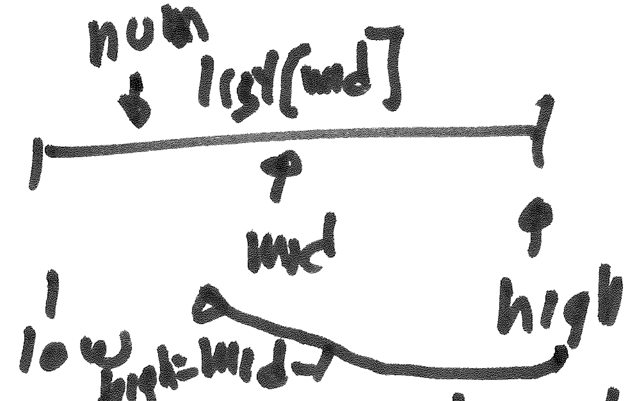
findPosAux(num, list, low, high)
Returns position of num
in list using boundaries
low and high

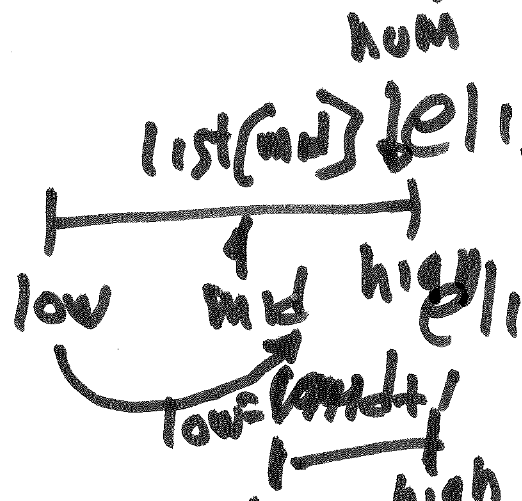
$$
\text{findPosAux}(num, list, low, high) = \begin{cases}
-1 & \text{if } high < low \text{ (end condition)} \\
mid = \dfrac{low + high}{2} & \\
\text{findPosAux}(num, list, low, mid-1) & \text{if } list[mid] > num \\
\text{findPosAux}(num, list, mid+1, high) & \text{if } list[mid] < num \\
mid & \text{if } list[mid] = num
\end{cases}
$$





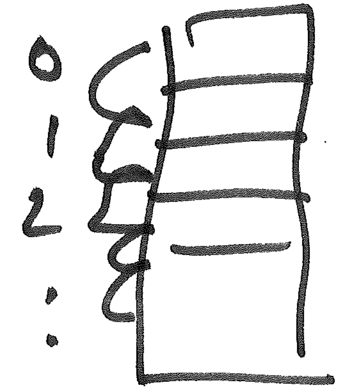
```
def findPosHelper(num, list, low, high):
    if high < low:
        return -1
    mid = (low+high)//2

    if list[mid] == num:
        return mid          //we found num at mid.
    elif list[mid] > num
        return findPosHelper(num, list, low, mid-1)
    elif list[mid] < num
        return findPosHelper(num, list, mid+1, high)
```






```
def findPos(num, list):
    return findPosHelper(num, list, 0, len(list)-1)
```

Implementation of a sequential search using recursion.



Without recursion

```
def findPos(num, list):
    for i in range(len(list)):
        if list[i] == num:
            return i

    return -1
```

With recursion

```
# find position of num in list starting at position startPos
def findPosHelper(num, list, startPos):
    #end condition
    if startPos == len(list):
        #search failed
        return -1
    if list[startPos] == num:
        #found num
        return startPos
    #Try search after startPos
    return findPosHelper(num, list, startPos+1)

def findPos(num, list):
    return findPosHelper(num, list, 0)
```

list

3	0
7	1
8	2 ← startPos = 2
4	3
1	4
9	5

len = 6

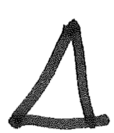
num = 9

a
↓
findPos(n, l) → 5
findPoshelper(n, l, 0)
findPoshelper(n, l, 1)
findPoshelper(n, l, 2)
⋮
findPoshelper(n, l, 5)
return 5

Turtle Graphics

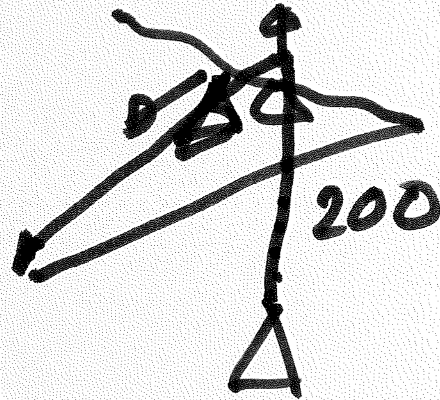
- Module in python that simplifies use of graphics.
- It was designed in the 70's to teach programming to childres. Used in a language called "Logo" and in Pascal.

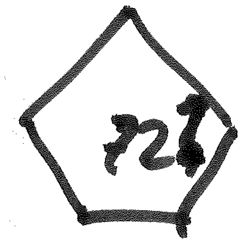
- Simple instructions


turtle

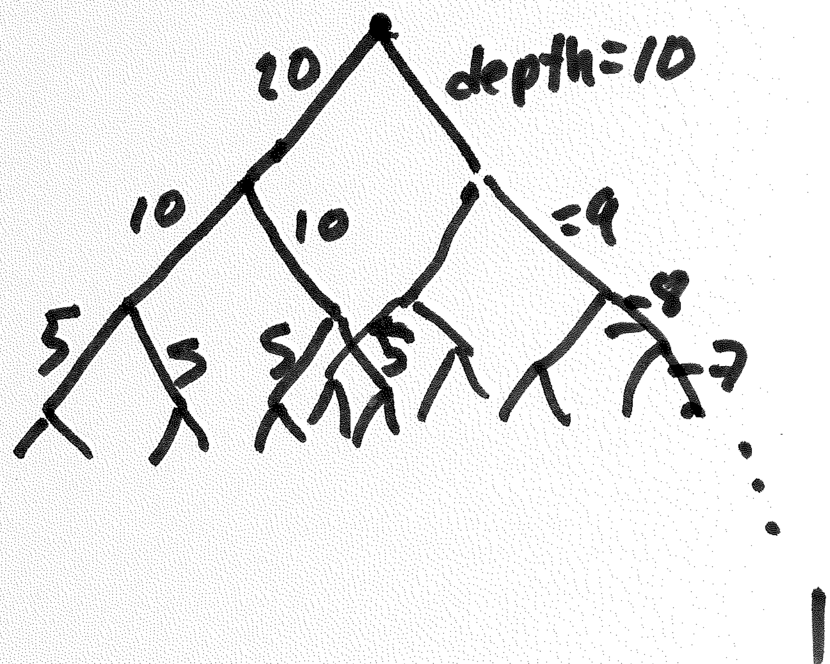
forward(n) — move forward n units
left(a) — turn left a degrees
right(a) — turn right a degrees

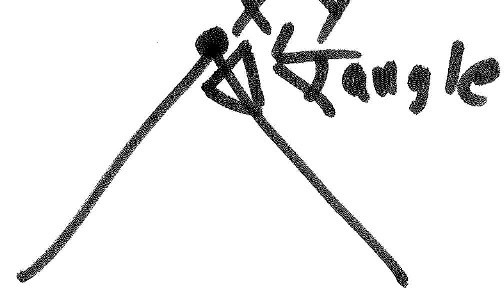
up() — pen up
down() — pen down.

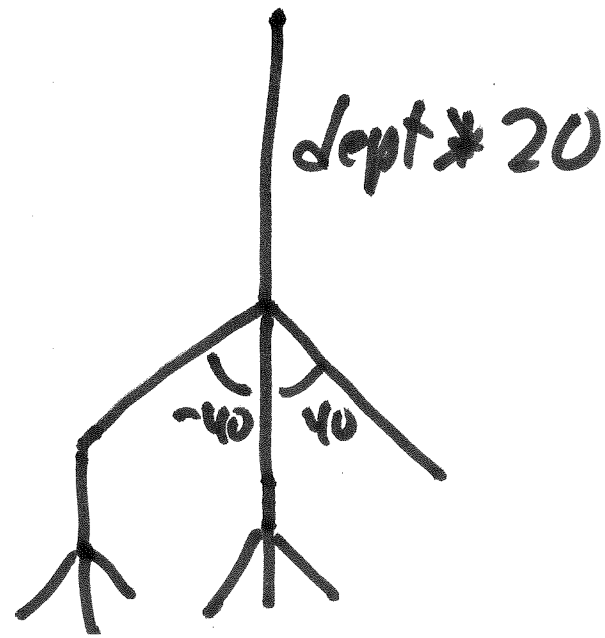




Recursive Figures and Fractuls







Numerical Analysis

It uses the computer to solve/approximate solutions of mathematical problems.

These methods can be used to solve problems in other branches of science.

- Numerical Derivative (approximate derivative,
- Numerical Integration
- Matrices
- Differential Equation approximation.

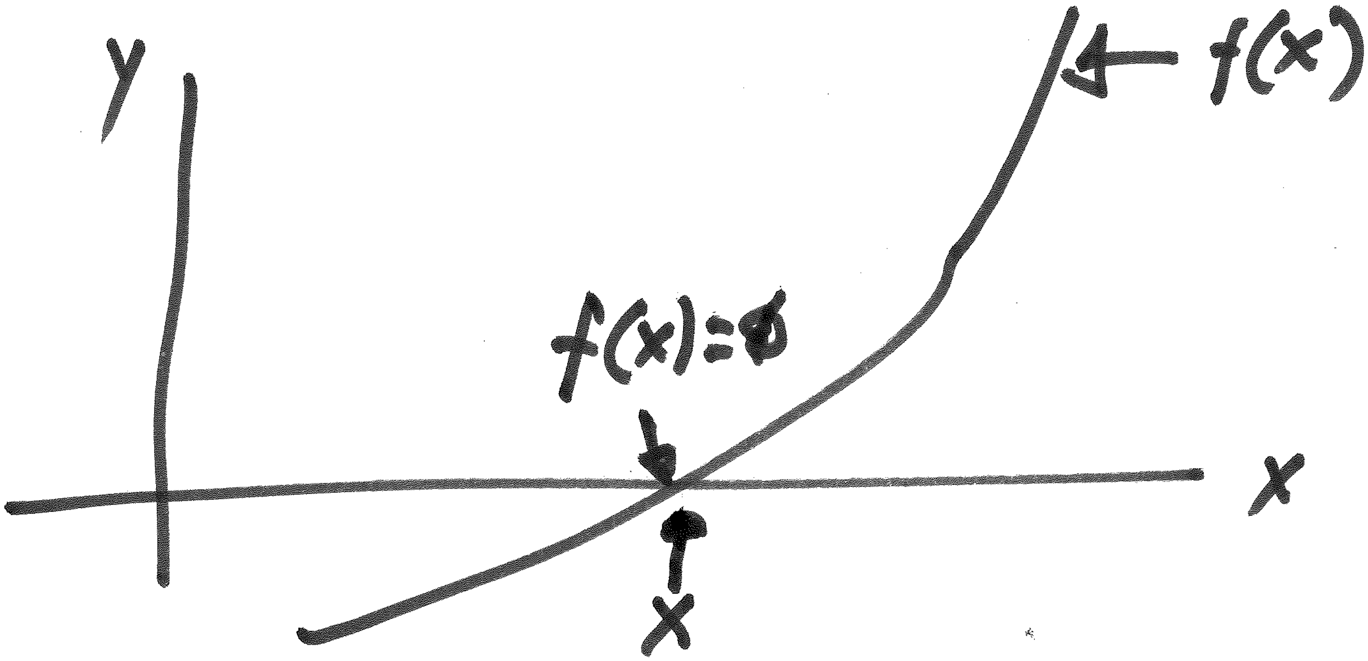
Example:

Obtain the solution of

$$f(x) = \phi$$

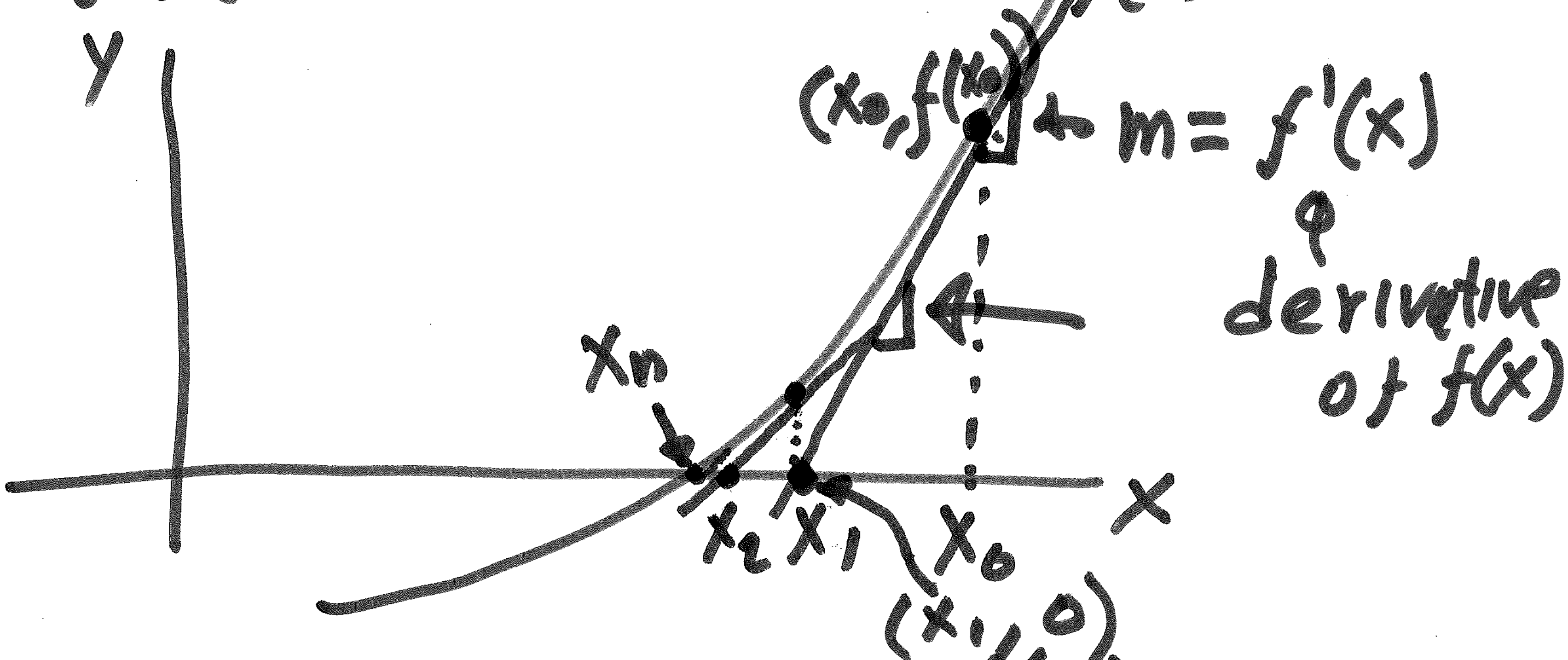
Finding a root of $f(x)$.

find x. suchth that $f(x) \cong \phi$



Newthon-Rapson method

It iteratively finds the solution of $f(x)=0$



- Build line tangent to $x_0, f(x_0)$
- where this line crosses X-axis that will be x_1

$$f(x_n) \approx \phi$$

- Building the tangent line

$$m = f'(x_0) = \frac{f(x_0) - \phi}{x_0 - x_1}$$

$$f'(x_0) = \frac{f(x_0)}{x_0 - x_1}$$

$$(x_0 - x_1)\, f'(x_0) = f(x_0)$$

$$x_0 - x_1 = \frac{f(x_0)}{f'(x_0)}$$

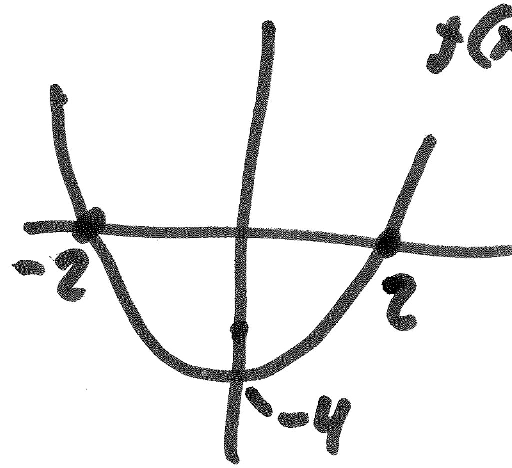
$$\boxed{x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}}$$

Newton-Rapson

Newthon.py

$f(x)=x^2-4=0$

(85) [redacted]



$x^2=4$

$x=\sqrt[2]{4}$

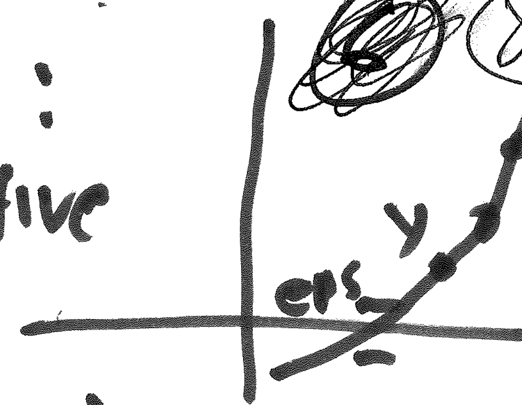
$x=+2,-2$

```
def f(x):
    return x**-4

def fp(x):
    return 2*x
```

$f'(x)=2x$

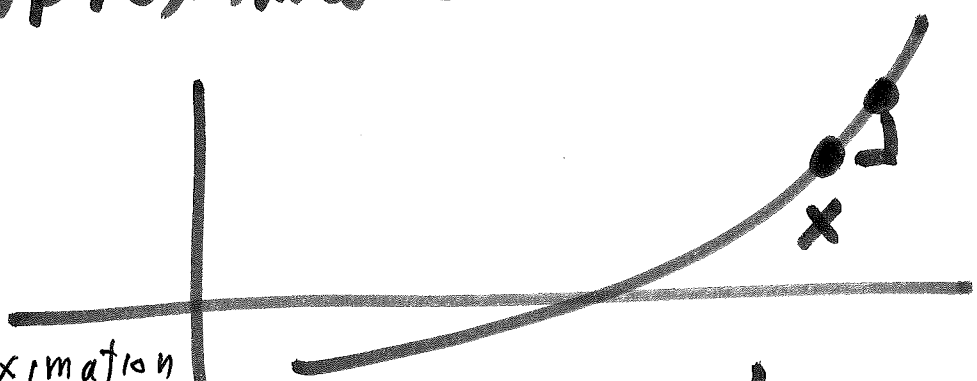
$x^3-2x+1=0$

```
def main():
    print("Newthon-Rapson").
    Xo=eval(input("Xφ="))
    eps=.00001 # Approx value |f(x)|<eps.
                 then stop.
    Y=1 #initialvalue for Y
    X=Xo #Initial value of X
```




```
    while abs(y) > eps:
        dy=fp(x)   #derivative
        y=f(x)
        x = x - y/dy
    print("result x=", x)
main()
```

Approximate derivative




```
#Use approximation
#of derivative.
def fp(x):
    eps=.00001
    #return 2*x
    return (f(x+eps)-f(x))/eps
```

$$f'(x) = \frac{\Delta y}{\Delta x}$$

$$\frac{dy}{dx} = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon)-f(x)}{\varepsilon}$$