Beam Programming Language:

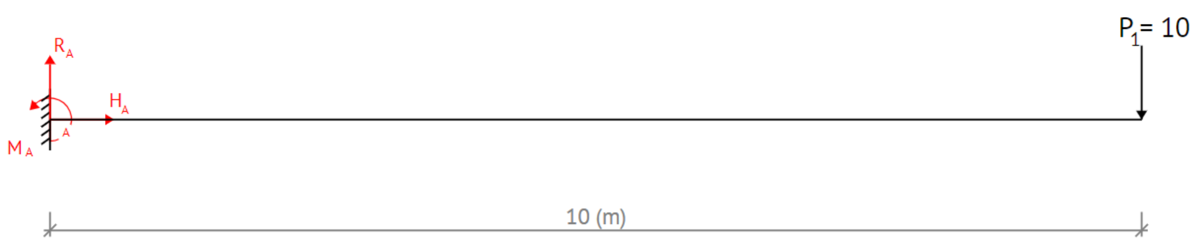
The Beam programming language is aimed at structural engineers who commonly work with beams. Beam allows engineers to define a beam's loading conditions quickly run calculations to find commonly needed values such as support reaction forces and internal shear and bending moment.

At its current stage of development, Beam only offers support for:

1. Cantilever Beams
2. Point loads at any angle
3. Rectangular Distributed Loads
4. Point Torques
5. Finding Resultant Forces
6. Finding Reaction Forces

Sample Programs:

The following beam will be represented in *Beam*.



Beam-Programming-Language > Simple1.beam

```
PointLoad Load{
    Magnitude -10   //The magnitude of the force in Newtons (N)
    Location 10     //distance from the y-axis in meters (m)
    Angle 90        //angle from the x-axis in degrees
}
```

This sample program is for a cantilever beam with a 10kg point load at a location 10meters from its support.

It outputs the following:

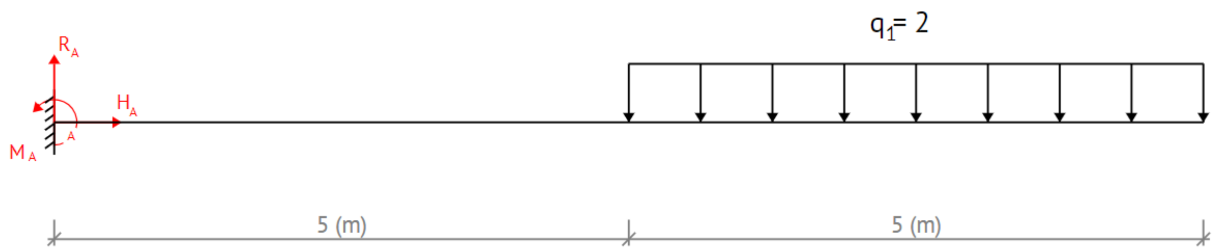
```
[Running] python -u "c:\Users\ausalsaka\Desktop\School\Adv Prog Lang\Beam-Programming-Language\Beam.py"
Calculating Resultant forces...
Resultant Forces: F_Rx= -0.0N  F_Ry= -10.0N  Moment= -100.0N*m

Calculating Reaction forces...
Reaction Forces: F_Ox= 0.0N  F_Oy= 10.0N  M_O= 100.0N*m

[Done] exited with code=0 in 1.141 seconds
```

Here is another simple program:

The following beam will represented in *Beam*.



Beam-Programming-Language > Simple2.beam

```
DistributedLoad DistL{
    Location 5
    Magnitude -2
    Length 5
}
```

This program is for a cantilever beam with a rectangular distributed load that starts at a distance 5 meters from the support and continues for another 5 meters. The load has a magnitude of 2 Newtons per meter (N/m).

It outputs the following:

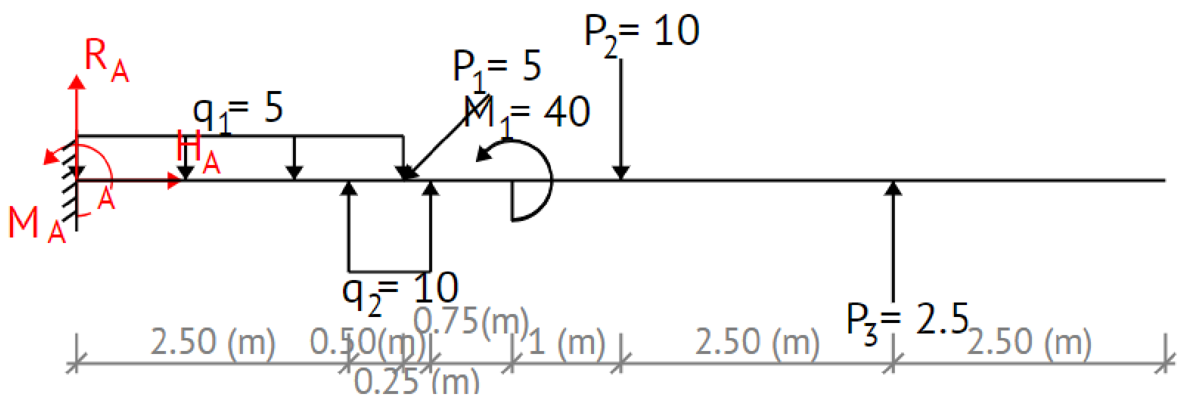
```
[Running] python -u "c:\Users\ausalsaka\Desktop\School\Adv Prog Lang\Beam-Programming-Language\Beam.py"
Calculating Resultant forces...
Resultant Forces: F_Rx= 0.0N  F_Ry= -10.0N  Moment= -75.0N*m

Calculating Reaction forces...
Reaction Forces: F_Ox= -0.0N  F_Oy= 10.0N  M_O= 75.0N*m

[Done] exited with code=0 in 1.091 seconds
```

Complex Program:

The following complexly loaded beam will be represented in *Beam*.



Beam-Programming-Language > Complex1.beam

```
PointLoad Load1 {
Magnitude -10.0
Location 5.0
Angle 90
}

PointLoad Load2{
    Magnitude -5
    Location 3
    Angle 45
}

PointLoad Load3{
    Magnitude 2.5
    location 7.5
    Angle 90
}

Torque Torque1{
    Magnitude 40
    Location 4

}

DistributedLoad DL1{
    Magnitude -5
    Length 3
    Location 0.0
}

DistributedLoad DL2{
    Magnitude 10
    Length .75
    Location 2.5
}
```

Here is the output:

```
[Running] python -u "c:\Users\ausalsaka\Desktop\School\Adv Prog Lang\Beam-Programming-Language\Beam.py"
Calculating Resultant forces...
Resultant Forces: F_Rx= -3.54N  F_Ry= -8.54N  Moment= -61.54N*m

Calculating Reaction forces...
Reaction Forces: F_Ox= 3.54N  F_Oy= 8.54N  M_O= 61.54N*m

[Done] exited with code=0 in 1.419 seconds
```

Beam was developed using textX and python.

Here is the .tx file containing the grammar of the language:

Beam-Programming-Language > Beam.tx

```
BeamModel:

    loads+=Load*;

Load:
    PointLoad | Torque | DistributedLoad;

PointLoad:
    "PointLoad" name=ID '{'
        properties+=Property
    '}';

Torque:
    "Torque" name=ID '{'
        properties+=Property
    '}';

DistributedLoad:
    "DistributedLoad" name=ID '{'
        properties+=Property
    '}';

Property:
    name=ID value=Value;

Value:
    FLOAT | STRING;


Comment:
  /\/\/.*$/
;
```

The following interpreter runs *Beam* according to the grammar from the .tx file.

```
from os.path import dirname, join
import numpy
import math

import textx;
from textx import metamodel_from_file
from textx.export import metamodel_export, model_export

def load_model(code):
    metamodel = textx.metamodel_from_file('Beam.tx')
    return metamodel.model_from_file(code)

# Define functions for beam calculations
def calculate_reaction_forces(model):
    # Placeholder function
    print("Calculating Resultant forces...")
    sigmaFx = 0.0
    sigmaFy = 0.0
    sigmaMoment = 0.0
    for load in model.loads:
        if load.__class__.__name__ == "PointLoad":
            magnitude = 0.0
            location = 0.0
            theta = 0.0
            for prop in load.properties:
                match prop.name:
                    case "Magnitude":
                        magnitude = prop.value
                    case "Location":
                        location = prop.value
                    case "Angle":
                        theta = prop.value
            Fx = magnitude * math.cos(math.radians(theta))
            Fy = magnitude * math.sin(math.radians(theta))
            moment = Fy * location
            sigmaFx = sigmaFx + Fx
            sigmaFy = sigmaFy + Fy
            sigmaMoment = sigmaMoment + moment
        if load.__class__.__name__ == "Torque":
            magnitude = 0.0
            location = 0.0
            for prop in load.properties:
                match prop.name:
```

```
                    case "Magnitude":
                        magnitude = prop.value
                    case "Location":
                        location = prop.value
            sigmaFy = sigmaFy + magnitude/location
        if load.__class__.__name__ == "DistributedLoad":
            length = 0.0
            height = 0.0
            location = 0.0
            for prop in load.properties:
                match prop.name:
                    case "Magnitude":
                        height = prop.value
                    case "Length":
                        length = prop.value
                    case "Location":
                        location = prop.value
            F_Ry = length * height
            moment = F_Ry * (location + length/2)
            sigmaFy += F_Ry
            sigmaMoment += moment

    print(f"Resultant Forces: F_Rx= {round(sigmaFx,2)}N  F_Ry= {round(sigmaFy,2)}N  Moment= {round(sigmaMoment,2)}N*m\n")

    print("Calculating Reaction forces...")
    F_Ox = -sigmaFx
    F_Oy = -sigmaFy
    M_O = -sigmaMoment

    print(f"Reaction Forces: F_Ox= {round(F_Ox,2)}N  F_Oy= {round(F_Oy,2)}N  M_O= {round(M_O,2)}N*m")




def calculate_shear_force(model, position):
    # Placeholder function
    print(f"Calculating shear force at position {position}...")

def calculate_bending_moment(model, position):
    # Placeholder function
    print(f"Calculating bending moment at position {position}...")



def main(debug=False):
```

```
    this_folder = dirname(__file__)

    #DOT Diagrams
    model = metamodel_from_file(join(this_folder, 'Beam.tx'), debug=False)
    metamodel_export(model, join(this_folder, 'beam.dot'))
    beam_model = model.model_from_file(join(this_folder, 'code.beam'))
    model_export(beam_model, join(this_folder, 'program.dot'))

    # Parse the code and perform beam calculations
    model = load_model('Complex1.beam')         #<-- modify this to choose a
sourcecode file
    calculate_reaction_forces(model)
    # calculate_shear_force(model, 2.5)         //<--to be implemented
    # calculate_bending_moment(model, 2.5)      //<--


if __name__ == "__main__":
    main()
```