KERBAL OPERATING SYSTEM 2020 useable terms.

Directions
R (pitch, yaw, roll).
Set mydir to R(a, b, c).

Print ship:direction:pitch.

Ship:direction:forevector

Ship:direction:topvector.

Ship:body:position.

Omg how to get a single coordinate out of a vector!

```
SET varname TO V(100,5,0).

varname:X.    // Returns 100.
V(100,5,0):Y. // Returns 5.
V(100,5,0):Z. // Returns 0.
```

Trajectories mod
print addons:tr:available
(this should print “true”).

How to set your target landing spot
set KSCLAUNCHPAD to latlng(-0.0971531143422002, -74.5576932487367).
Addons:tr:settarget to KSCLAUNCHPAD

To check this,
Print addons:tr:hastarget
(this should print “true”).

Warpto.
This is to instigate warp.

Eta:apoapsis

Obt is orbit

mapview

Return

Body:geopositionof
North:vector
Up:vector
Vecdraw
a:altitudepostition(a:terrainheight).

When distancetoground() < 500 then { gear on. }

Function distancetoground {
Return Altitude - body:geopositionof(ship:position):terrainheight.
}

UNTIL loops are used for a loop that end when some condition becomes true.
FOR loops are used to itterate over the items in a list.
FROM loops are for when you want a fixed number of iterations of the loop.

so for this type of case UNTIL would likely be best
Just remember to have a WAIT 0. somewhere in the UNTIL loop

```
    if stage:liquidfuel > 1 {

print stage:resources[0].
    print stage:liquidfuel.
    ship:parts[67]:getmodule("moduleresourcedrain"):setfield("drain",true).

wait 0.1.

}
```

```
    if stage:liquidfuel > 1 {

print stage:resources[0].
    print stage:liquidfuel.
    ship:parts[67]:getmodule("moduleresourcedrain"):setfield("drain",false).

wait 0.1.

}
```

To see resources.
Print stage:resources.
Print stage:resources[0].
Print stage:resources[1].
[0] liquidfuel
[1] oxidiser

To transfer fuel.

```
SET foo TO TRANSFERALL("OXIDIZER", SHIP:PARTS[0], SHIP:PARTS[1]).
SET foo:ACTIVE to TRUE.
```

Using Fuel Drain.
Moduleresourcedrain
[0]resources
[1] drain rate
[2] drain (is boolean)
[3] drain mode
[4] drain (is ksp action)
[5] stop draining (is kspaction)
[6] toggle draining (is kspaction)
[7] toggle resource drain mode (is kspaction).

ship:parts[67]:getmodule(“moduleresourcedrain”):setfield(“drain”,true).
ship:parts[67]:getmodule(“moduleresourcedrain”):setfield(“drain”,true).

TO CALL A PART FROM AN ASSIGNED NAME!!!
ship:partstagged(“ string name “)[0].

ship:partstagged(“robotty.23.stuff”)[0]:getmodule(“moduleroboticrotationservo”):setfield(‘targetangle”90).

Strings need to be put in brackets with quotes.
Wait until ship:status = landed. //will return an error
Wait until ship:status = (“landed”). // will work.

Word to remember - preserve.
To keep a loop running and checking.
This will create a constant background check.

To use the vessel structure.
Print ship.
Set nome to ship.
Print nome.
This should return - VESSEL(“aa_pete_booster_only_v2”).
Nome:verticalspeed.

Staging should use a “When”, not an until.

```
WHEN MAXTHRUST = 0 THEN {
    PRINT "Staging".
    STAGE.
    PRESERVE.
}.
```

WAIT UNTIL, will pause the script and keep it from exiting until the conditions are met.

UNTIL loop

```
SET MYSTEER TO HEADING(90,90). //90 degrees east and pitched up 90 degrees (straight up)
```

```
LOCK STEERING TO MYSTEER. // from now on we'll be able to change steering by just assigning a new value to MYSTEER

UNTIL APOAPSIS > 100000 {

    SET MYSTEER TO HEADING(90,90). //90 degrees east and pitched up 90 degrees (straight up)    }.
```

This loop will continue to execute all of its instructions until the apoapsis reaches 100km. Once the apoapsis is past 100km, the loop exits and the rest of the code continues.

ELSE IF

```
} ELSE IF SHIP:VELOCITY:SURFACE:MAG >= 100 AND SHIP:VELOCITY:SURFACE:MAG < 200 {
```

Each time this loop iterates, it will check the surface velocity. If the velocity is below 100m/s, it will continuously execute the first block of instructions. Once the velocity reaches 100m/s, it will stop executing the first block and start executing the second block, which will pitch the nose down to 80 degrees above the horizon.

PRINT AT

```
PRINT "Pitching to 80 degrees" AT(0,15).

        PRINT ROUND(SHIP:APOAPSIS,0) AT (0,16).
```

IF statements.

```
set x to 1.
```

```
if x = 1 {

  print "x is one".

}
```

Or

```
if x = 1

  print "x is one".
```

Equals to or bigger than:

```
1 >= 1

2 >= 1
```

Equals to or smaller than

```
1 <= 1

1 <= 2
```

Is not equal to:

```
1 <> 2
```

How to list the parts on your ship.

Enter
List parts.

This should show you a list of the parts on the ship.

OPEN TERMINAL INSTRUCTIONS
When creating programs within the terminal.
Enter
Edit enter the name of your program.

Heres an example -
Edit demo5.

This will open the screen underneith and call your new program demo5.

LIST
This will list all your current programs saved on the kos internal drive on the rocket.

Switch to 0.

List
This will list all the programs in the scripts menu.
List archives. Will show the different folders where programs are stored.

Using Boot files to launch files (especially for debugging)
Create Boot file to load in KOS computer.
Runpath (“0:/debug.ks”).

This will run our current file at the launch pad, so we can revert to launch without having to revert to the VAB to reboot the files every time.

Debugging trick

When encountering error messages, to simplify the code, ie.

UNTIL APOAPSIS > 100 {
Comment out the line ie. //UNTIL APOAPSIS >100{
And enter -
UNTIL TRUE {
Then you can see if the error message happens after the command or before.

As a quirk of KOS, if there is multiple lines of code that are the same, ie.
Stage.
Stage.
It takes a physics tick to acknowledge the 1st stage command, therefore not ackowleding the 2nd stage command.
So a WAIT command it required.
STAGE.
WAIT 1.0.
STAGE.

WAIT UNTIL FALSE.
Can be added at the end of a program to stop it from terminating and resetting the controls to zero, for manual control.

Other commands that might be useful
Ship:availablethrust.
Declare global.
Ctrl C can kill the program in the command window.

LOCKing a variable means the computer will constantly look for updates.
SET will make the variables value the same throughout the program without updating it.

Staging program.
STAGE:READY (useful command).

FUNCTION dosafestage {
Wait until stage:ready.
Stage.
}

Then when calling that function type
dosafestage().

```
FUNCTION doautostage {
        IF NOT(DEFINED oldthrust) {
        DECLARE GLOBAL oldthrust to SHIP:AVAILABLETHRUST.
        }
        IF SHIP:AVAILABLETHRUST <oldthrust - 10) {
        dosafestage(). Wait 1.
        Set oldthrust to shipavailablethrust
        }
        }
Until apoapsis >100000 {
        doautostage().
}

SET oldthrust to SHIP:AVAILABLETHRUST.
UNTIL APOAPSIS > 100000 {
PRINT “available:  “ + SHIP:AVAILABLETHRUST.
Print “old: “ + oldthrust.
IF SHIP:AVAILABLETHRUST < (oldthrust - 10) {
Stage. Wait 1.
Set oldthrust to SHIP:AVAILABLE:THRUST.
}
{
```

Parts action window editing

Hinge.03-941066
uid=1440671744

```
SET myPart TO SHIP:PARTSDUBBED("my nametag here")[0].
SET myPart TO SHIP:PARTSDUBBED("hinge.03")[0].

// Only if you expected to get
// exactly 1 such part, no more, no less.
SET myPart TO SHIP:PARTSDUBBED("my nametag here")[0].

// Handling the case of more than
// one part with the same nametag,
// or the case of zero parts with
// the name:
SET allSuchParts TO SHIP:PARTSDUBBED("my nametag here").
```

Print ship:partsdubbed(“hinge.03”).

Print ship:partsnamed(“hinge.03”).

ship:partsnamed(“hinge.03”):setfield(“targetangle”,90).

ship:partsnamed(“hinge.03”):setfield(“targetangle”,90).

PART(HINGE.04,UID=2555234304).

HOW TO DO IT PROPERLY!!!

List parts in plist.
Print plist[0].
Set rootpart to plist[0].
Print rootpart:modules.
Set mods to rootpart:modules.
Print mods[0].

Set mods to rootpart:getmodule(“modulesas”).
Print mod.

Set mod to rootpart:getmodule(“modulecommand”).
Print mod.

Print mod:allfields.

Print mod:allactions

Print mod:allevents.

Set capevents to mod:allevents.
Print capevents[0].

Print capevents[1].

mod :doevent(“rename-vessel”).

HOW TO EXTEND AN ANTENNA

List parts in plist.
Print plist.
Set antdep to plist[14].
Print antdep:modules.

Set mods to antdep:modules.
Print mods.
Set mod to antdep:getmodule(“moduledeployableantenna”).
Print mod.
mod:doevent(“extend antenna”).

CAN IT BE SIMPLIFIED?

List parts in plist.
Print plist.
Set antdep to plist[14].
Print antdep:modules.
Set mods to antdep:getmodule(“moduledeployableantenna”).
Print mods.
mods:doevent(“extend antenna”).

How to move a hinge!

// how to move a hinge using KOS and the PAW (part action group)
// thanks to nuggreat for help on this! It wasnt easy!
// here is a perfect video that explains it step by step, watch from 11mins in.
// https://www.youtube.com/watch?v=3bujGZP7a_o&feature=youtu.be

List parts in plist. // this lists all the parts on the ship and gives them a number
Print plist. //find the number of the part you want to control in the []
Set hangle to plist[12]. //enter the number of the part into the brackets []
Print hangle:modules. //hangle is a variable name i made up, ie hinge angle
Set mods to hangle:getmodule(“moduleroboticservohinge”).
Print mods.
mods:setfield(“target angle”,90). //set field is used as it has a range from 0 to 180.

To add a variable angle.

Set x to 10.
mods :setfield(“target angle”,

Target angle [0]
[0]moduleroboticservohinge.

partlist[12]:getmodule(“moduleroboticservohinge”):setfield(“target angle”,20).
Ship:parts[12].

Ship:parts[12]:getmodule(“moduleroboticservohinge”):setfield(“target angle”,20).

```
Ship:parts[12]:getmodule("moduleroboticservohinge"):setfield("target
angle",99).
```

Ship:parts[12]:getmodule("moduleroboticservohinge"):setfield("target angle",99).

Docking port is [18].
ModuleDockingNodem [0].
Control from here is [6].

Airbrakes.
Module
[0] “moduleaerosurface”
[1] “fxmodulelookatconstraint”

Moduleaerosurface.
[0] authority limiter
[1] deploy angle
[2] pitch
[3] yaw
[4] deploy
[5] toggle
[6] toggle deploy
[7] extend
[8] retract

Ship:parts[13]:getmodule("Moduleaerosurface"):setfield("pitch",true).
Ship:parts[13]:getmodule("Moduleaerosurface"):setfield("yaw",true).

Setting pitch.

Print ship:control:rotation

Print ship:control:translation

Print ship:control:pitch

Vectors are exactly what you need. They define a direction in 3-dimensional space. I went through exactly the same thing a while back. I had to write notes.

kOS provides some builtin directions that you can derive other info from. I think of the directions as little "aircraft" that fly along with your own aircraft or rocket, just like the little "aircraft" in the navball. Once I get that image in my head, I can do the trigonometry myself or find it on the Internet.

FACING (ship:facing) is a direction that matches your vessel. It is the "aircraft" on the navball. But it's vectors make no sense unless they are compared to other, known vectors that remain fixed relative to your vessel's position.

UP (ship:up) is a direction like an "aircraft" with it's nose pointing straight up and its roof pointing north. It tells you where your vessel's zenith is at all times.

NORTH (ship:north) is a direction like an "aircraft" with it nose pointing north and it's roof pointing straight down. It tells you where your vessel's north is at all times (except right on the north or south pole...).

So your pitch is just the angle between the vessel's zenith and where the nose of your vessel is pointing. Because the convention is to define pitch as the number of degrees above the vessel "horizon", you subtract it from 90:

set currentpitch to 90 - vectorangle(up:vector, ship:facing:forevector).

To get a useable pitch for reentry.

```
clearscreen.
until ship:altitude >20000 {
set currentpitch to vectorangle(up:vector, ship:facing:forevector).
print "pitch:     " + round(currentpitch, 1) + "    " at(5, 5).
wait 0.1.
}
```

FRONT FLAP CONTROL

```
clearscreen.
until ship:altitude > 200000  {
```

set currentpitch to vectorangle(up:vector, ship:facing:forevector).
print "pitch: " + round(currentpitch, 1) + "   " at(5, 5).
set btrt to 90.
set btlt to 90.
Ship:parts[9]:getmodule("moduleroboticservohinge"):setfield("target angle",btrt).
Ship:parts[11]:getmodule("moduleroboticservohinge"):setfield("target angle",btlt).
Ship:parts[25]:getmodule("moduleroboticservohinge"):setfield("target angle",currentpitch).
Ship:parts[27]:getmodule("moduleroboticservohinge"):setfield("target angle",currentpitch).
wait 0.1.
}

Landing on a geoposition.

Print ship:geoposition.
Ksc -0.0971531143422002
    -74.5576932487367

```
SET spot to SHIP:GEOPOSITION.
```

Ship:facing.

Lock steering to ksclaunchpad:altitudeposition(100000).

```
set KSCLAUNCHPAD to latlng(-0.0972092543643722, -74.557706433623).  //The launchpad at
the KSC
```

Getcoordinates.
Unlock steering
Unlock throttle.

Lock steering to kill, // resets the steering back the last heading.

set Ksclat to -0.0971531143422002.
set Ksclng to -74.5576932487367.

ship:direction.
steering manager:resettodefault().

PRINT ALLWAYPOINTS().

```
SET spot TO LATLNG(10, 20).      // Initialize point at latitude 10,
                                 // longitude 20
```

```
PRINT spot:LAT.                 // Print 10
PRINT spot:LNG.                 // Print 20

PRINT spot:DISTANCE.            // Print distance from vessel to x
PRINT spot:HEADING.             // Print the heading to the point
PRINT spot:BEARING.             // Print the heading to the point
                                // relative to vessel heading
```

Might be handy

```
//the following are all vectors,
mainly for use in the roll, pitch,
and angle of attack calculations
                                    lock rightrotation to
ship:facing*r(0,90,0).
   lock right to rightrotation:vector.
//right and left are directly along wings
   lock left to (-1)*right.

   lock up to ship:up:vector. //up and down
are skyward and groundward
   lock down to (-1)*up.

   lock fore to ship:facing:vector. //fore
and aft point to the nose and tail
   lock aft to (-1)*fore.

   lock righthor to vcrs(up,fore). //right
and left horizons
   lock lefthor to (-1)*righthor.

   lock forehor to vcrs(righthor,up).
//forward and backward horizons
   lock afthor to (-1)*forehor.

   lock top to vcrs(fore,right). //above the
cockpit, through the floor
   lock bottom to (-1)*top.
```

```
   //the following are all angles, useful for
control programs
   lock absaoa to
vang(fore,srfprograde:vector). //absolute
angle of attack
   lock aoa to
vang(top,srfprograde:vector)-90. //pitch
component of angle of attack
   lock sideslip to
vang(right,srfprograde:vector)-90. //yaw
component of aoa
   lock rollangle to
vang(right,righthor)*((90-vang(top,righthor))
/abs(90-vang(top,righthor))). //roll angle, 0
at level flight
   lock pitchangle to
vang(fore,forehor)*((90-vang(fore,up))/abs(90
-vang(fore,up))). //pitch angle, 0 at level
flight
   lock glideslope to
vang(srfprograde:vector,forehor)*((90-vang(sr
fprograde:vector,up))/abs(90-vang(srfprograde
:vector,up))).
```