Implementing the Gram-Schmidt Algorithm

Let's implement the notions of **orthogonality** and the **Gram-Schmidt algorithm**.
This is a very nice example of the beauty and power of **functional programming**.

Orthogonality

Let's implement the notion of an orthogonal list of vectors.

```
orthogonalQ::Usage =
  "orthogonalQ[vs_?MatrixQ] returns True if the vectors in the list of
    vectors vs are mutually orthogonal, and returns False otherwise.";
```

(* Inductive definition:
A list containing one vector is orthogonal.
A list containing more than one vector is orthogonal if
the first vector in the list is orthogonal to the remaining vectors, and
removing the first vector from the list results in a (shorter) list of vectors which are mutually orthogonal.
*)

```
orthogonalQ[vs_?MatrixQ] :=
 If[Length[vs] == 1, True,
  orthogonalQ[Rest[vs]] && orthogonalQ[Rest[vs], First[vs]]]

orthogonalQ[vs_?MatrixQ, v_?ListQ] :=
 If[Length[vs] == 1, orthogonal2Q[First[vs], v],
  orthogonalQ[Rest[vs], v] && orthogonal2Q[First[vs], v]]

orthogonal2Q[u_?ListQ, v_?ListQ] := If[u.v == 0, True, False]
```

Testing.

```
v1 = {1, 0, 0};
v2 = {0, 1, 0};
v3 = {0, 0, 1};
v4 = {2, 3, 4};

orthogonal2Q[v1, v2]
orthogonal2Q[v3, v4]

True
```

```
False
```

```
orthogonalQ[{v1, v2, v3}]
orthogonalQ[{v1, v2, v4}]

True
```

```
False
```

Projection of a vector onto a subspace spanned by an orthogonal basis

Let's implement the notion of the **projection** of a vector onto a subspace which is spanned by an orthogonal basis.

```
proj::Usage = "proj[us,u] returns the projection of the vector
    u onto the subspace spanned by the orthogonal basis us.";
```

```
proj[us_?orthogonalQ, u_?ListQ] :=
 Module[{u1 = u.us[[1]] / us[[1]].us[[1]] us[[1]]},
  If[Length[us] == 1, u1,
   u1 + proj[Rest[us], u]]]
```

Testing.

```
proj[{v1}, v4]
proj[{v2}, v4]
proj[{v3}, v4]
```

```
{2, 0, 0}
```

```
{0, 3, 0}
```

```
{0, 0, 4}
```

```
proj[{v1, v2}, v4]
proj[{v1, v3}, v4]
proj[{v2, v3}, v4]
```

```
{2, 3, 0}
```

```
{2, 0, 4}
```

```
{0, 3, 4}
```

```
proj[{v1, v2, v3}, v4]
```

```
{2, 3, 4}
```

Gram-Schmidt

Let's implement the **Gram-Schmidt** algorithm.

```
gramSchmidt::Usage =
  "gramSchmidt[vs] returns an orthogonal list of vectors, us, with
    the property that the first k vectors in vs have the same
    span as the first k vectors in us, for 1<=k<=Length[vs].";
```

```
gramSchmidt[vs_?MatrixQ] :=
 If[Length[vs] == 1, vs,
  Module[{us = gramSchmidt[Drop[vs, -1]], u = vs[[-1]]},
   Append[us, u - proj[us, u]]]]
```

Testing.

```
v1 = {1, 1 / 2, 0};
v2 = {1, 2, 0};
v3 = {1, 2, 3};
vs = {v1, v2, v3};

us = {u1, u2, u3} = gramSchmidt[vs];
```

Display the results

The Arrow3D package can be downloaded from Wolfram Research.
showColorful3DVectors is based on a similar procedure written by Selwyn Hollis.

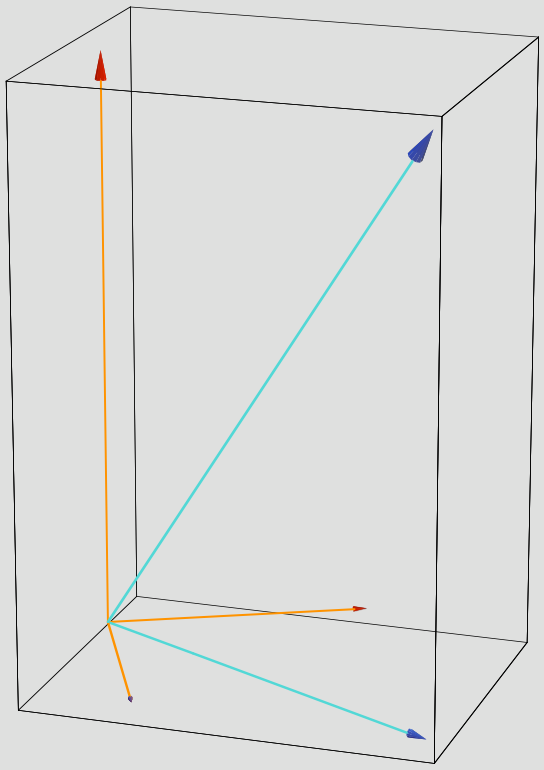
```
<< Arrow3D`Arrow3D`
```

```
showColorful3DVectors[vecs_, shaftColor_, headColor_, opts___] :=
 Show[Graphics3D[Flatten[{Thickness[.004], shaftColor,
     Table[Arrow3D[vecs[[i, 1]],
       vecs[[i, 1]] + vecs[[i, 2]], HeadColor → headColor],
      {i, Length[vecs]}]}]],
  opts, ViewPoint → {6, 2, 2} ]
```

```
<< Graphics`Graphics`
```

```
o = {0, 0, 0};

DisplayTogether[
  vPlot = showColorful3DVectors[{{o, v1}, {o, v2}, {o, v3}},
    MediumTurquoise, RoyalBlue],
  uPlot = showColorful3DVectors[{{o, u1}, {o, u2}, {o, u3}},
    Orange, Red]];
```



Where is the third blue vector?