Mike Walker - Term Project

In [1]:
```
import seaborn as sns
import numpy as np
import PIL
import pathlib

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import regularizers
import tensorflow.keras.layers.experimental.preprocessing as tfe
from tensorflow.keras.models import Sequential

print(tf.__version__)
```

```
2.6.0
```

In [2]:
```
def show_picture(noun):
    preview_image = list(data_dir.glob(noun + '/*'))
    preview_image_url = str(preview_image[0])
    display(PIL.Image.open(preview_image_url))
```

Activate GPU

In [3]:
```
#Activate and select the GPU
my_desktop_gpus = tf.config.experimental.list_physical_devices('GPU')
if len(my_desktop_gpus) > 0:
    for gpu in my_desktop_gpus:
        tf.config.experimental.set_memory_growth(gpu, False)
        print(f'{gpu} memory growth: {tf.config.experimental.get_memory_growth(gpu)
```

```
PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU') memory growth: Fals
e
```

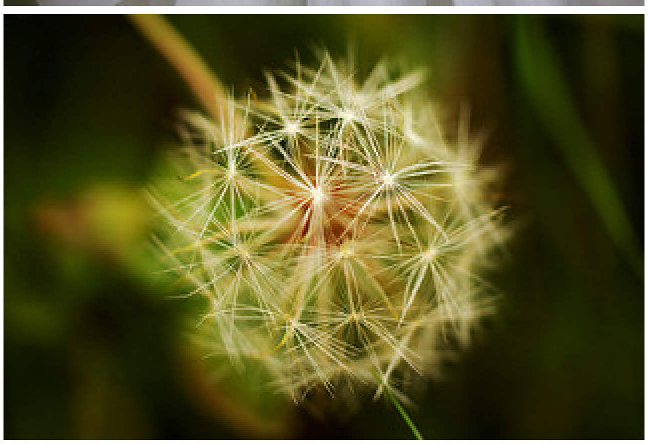
Select Photo Set

In [4]:
```
test_examples = {
    "family_padded": r"padded_family_photos",
    "family_no_pad": r"family_photos",
    "family_tiny_pad": r"padded_portrait_family",
    "flowers": r"flower_photos"
}

folder_url = test_examples["flowers"]
print(folder_url)

data_dir = pathlib.Path(folder_url)

image_count = len(list(data_dir.glob('*/*')))
print(image_count)
```

```
flower_photos
3665
```

Test Photo Set Loaded Correctly

In [5]:
```
if (folder_url == test_examples["flowers"]):
    show_picture("daisy")
    show_picture("dandelion")
    show_picture("roses")
    show_picture("sunflowers")
    show_picture("tulips")
else:
    show_picture("landon")
    show_picture("mike")
    show_picture("robin")
```



In [6]:
```
### Set image and model parameters
```

In [7]:
```
if (folder_url == test_examples["family_tiny_pad"]):
    batch_size = 4 #had to adjust for large images
    img_height = 2049
    img_width = 1539
elif (folder_url == test_examples["flowers"]):
    batch_size = 64
```

```
    img_height = 240
    img_width = 180
elif (folder_url == test_examples["family_no_pad"]):
    batch_size = 4
    img_height = 2048 #not actual size, just reduced it
    img_width = 1536  #not actual size, just reduced it
elif (folder_url == test_examples["family_padded"]):
    #This example has a massive padding.
    batch_size = 4 #had to adjust for large images
    img_height = 2305
    img_width = 2100


N_TRAIN = int(1e4)
STEPS_PER_EPOCH = N_TRAIN//batch_size

num_of_epochs = [60]
```

Load Training Dataset and Validation Dataset

In [8]:
```
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 3665 files belonging to 5 classes.
Using 2932 files for training.
```

In [9]:
```
val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 3665 files belonging to 5 classes.
Using 733 files for validation.
```

Verify Output and Shape

In [10]:
```
class_names = train_ds.class_names
print(class_names)
```

```
['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']
```

In [11]:
```
for image_batch, labels_batch in train_ds:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
```

```
(64, 240, 180, 3)
(64,)
```

Setup and Run Model

In [12]:
```
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

In [13]:
```
normalization_layer = tfe.Rescaling(1./255)
```

In [14]:
```
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
# Notice the pixel values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image))
```

```
0.0 1.0
```

In [15]:
```
# Improving the learning model by introducing variations in the training data
data_augmentation = Sequential([
    tfe.RandomFlip("horizontal", input_shape=(img_height, img_width, 3)),
    tfe.RandomRotation(0.1),
    tfe.RandomZoom(0.1),
])
```

In [16]:
```
#model inspried by: https://www.tensorflow.org/tutorials/keras/overfit_and_underfit
num_classes = len(class_names) #outputs

model = Sequential([
  data_augmentation,
  tfe.Rescaling(1./255),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes, name="outputs")
])

#epochs are high (50 and in one case 500) so adding this to help improve the conver
#by gradually reducing the learning rate as training progresses (as stated in Tenso
#from what I understood, it help led to a more stable/faster convergence to a good
lr_schedule = tf.keras.optimizers.schedules.InverseTimeDecay(
  0.001,
  decay_steps=STEPS_PER_EPOCH*1000,
  decay_rate=1,
  staircase=False)

#Selected Adam for the optimizer
```

```
optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule)


model.compile(optimizer=optimizer,
                  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
                  metrics=['accuracy'])

model.summary()

history = model.fit(train_ds, validation_data = val_ds, epochs = num_of_epochs[0])

results_data = {'Training Accuracy': history.history['accuracy'], 'Validation Accur

history = None
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential (Sequential)      (None, 240, 180, 3)       0
_________________________________________________________________
rescaling_1 (Rescaling)      (None, 240, 180, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 240, 180, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 120, 90, 16)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 120, 90, 32)       4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 60, 45, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 60, 45, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 22, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 30, 22, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 42240)             0
_________________________________________________________________
dense (Dense)                (None, 128)               5406848
_________________________________________________________________
outputs (Dense)              (None, 5)                 645
=================================================================
Total params: 5,431,077
Trainable params: 5,431,077
Non-trainable params: 0
_________________________________________________________________
Epoch 1/60
46/46 [==============================] - 5s 33ms/step - loss: 1.6113 - accuracy: 0.2
797 - val_loss: 1.2949 - val_accuracy: 0.4502
Epoch 2/60
46/46 [==============================] - 1s 17ms/step - loss: 1.2521 - accuracy: 0.4
894 - val_loss: 1.0494 - val_accuracy: 0.5771
Epoch 3/60
46/46 [==============================] - 1s 17ms/step - loss: 1.0589 - accuracy: 0.5
754 - val_loss: 1.0028 - val_accuracy: 0.6221
Epoch 4/60
46/46 [==============================] - 1s 17ms/step - loss: 0.9670 - accuracy: 0.6
211 - val_loss: 0.9947 - val_accuracy: 0.6221
Epoch 5/60
46/46 [==============================] - 1s 17ms/step - loss: 0.9057 - accuracy: 0.6
429 - val_loss: 0.8860 - val_accuracy: 0.6658
Epoch 6/60
46/46 [==============================] - 1s 17ms/step - loss: 0.8618 - accuracy: 0.6
675 - val_loss: 0.9322 - val_accuracy: 0.6576
Epoch 7/60
46/46 [==============================] - 1s 17ms/step - loss: 0.8143 - accuracy: 0.6
695 - val_loss: 0.8531 - val_accuracy: 0.6985
Epoch 8/60
46/46 [==============================] - 1s 17ms/step - loss: 0.7850 - accuracy: 0.6
883 - val_loss: 0.8606 - val_accuracy: 0.6767
```

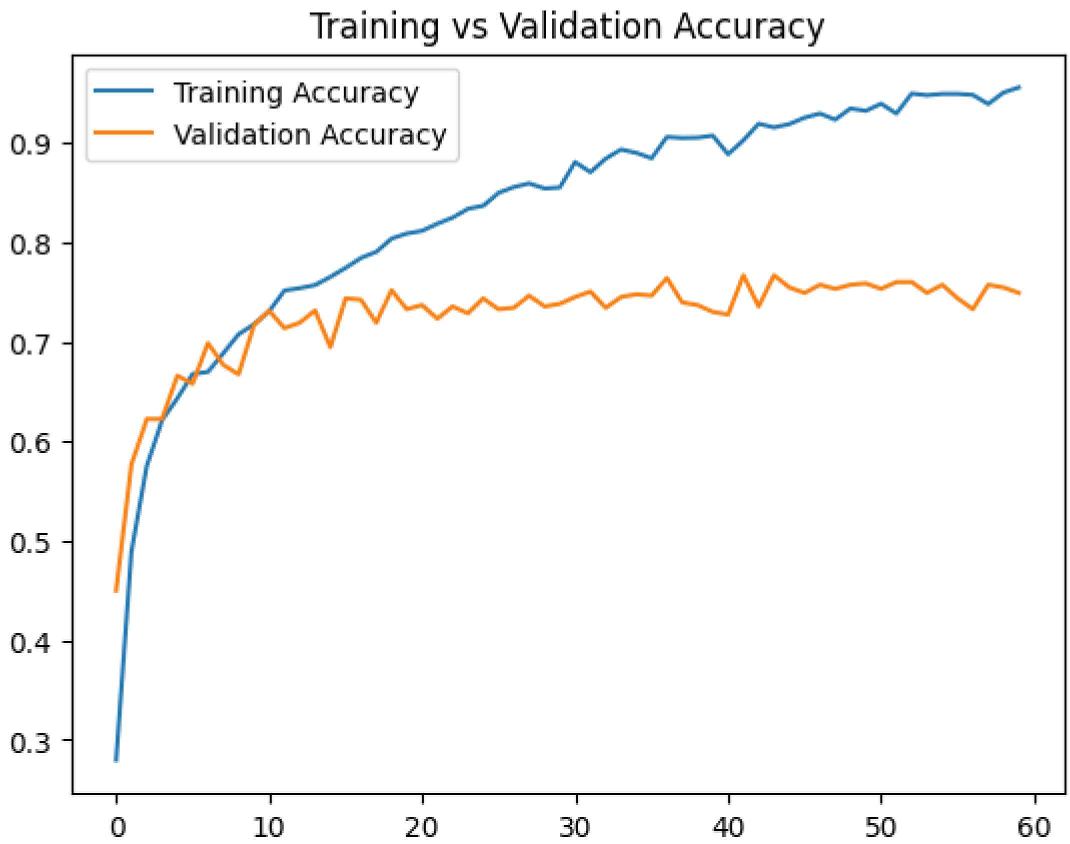
```
Epoch 9/60
46/46 [==============================] - 1s 17ms/step - loss: 0.7729 - accuracy: 0.7
074 - val_loss: 0.8477 - val_accuracy: 0.6671
Epoch 10/60
46/46 [==============================] - 1s 17ms/step - loss: 0.7278 - accuracy: 0.7
173 - val_loss: 0.7805 - val_accuracy: 0.7162
Epoch 11/60
46/46 [==============================] - 1s 17ms/step - loss: 0.6880 - accuracy: 0.7
306 - val_loss: 0.7422 - val_accuracy: 0.7312
Epoch 12/60
46/46 [==============================] - 1s 17ms/step - loss: 0.6501 - accuracy: 0.7
514 - val_loss: 0.8251 - val_accuracy: 0.7135
Epoch 13/60
46/46 [==============================] - 1s 17ms/step - loss: 0.6466 - accuracy: 0.7
538 - val_loss: 0.7954 - val_accuracy: 0.7190
Epoch 14/60
46/46 [==============================] - 1s 17ms/step - loss: 0.6329 - accuracy: 0.7
568 - val_loss: 0.7590 - val_accuracy: 0.7312
Epoch 15/60
46/46 [==============================] - 1s 17ms/step - loss: 0.6127 - accuracy: 0.7
650 - val_loss: 0.8522 - val_accuracy: 0.6944
Epoch 16/60
46/46 [==============================] - 1s 17ms/step - loss: 0.6106 - accuracy: 0.7
742 - val_loss: 0.7598 - val_accuracy: 0.7435
Epoch 17/60
46/46 [==============================] - 1s 17ms/step - loss: 0.5536 - accuracy: 0.7
841 - val_loss: 0.7441 - val_accuracy: 0.7422
Epoch 18/60
46/46 [==============================] - 1s 17ms/step - loss: 0.5514 - accuracy: 0.7
902 - val_loss: 0.7596 - val_accuracy: 0.7190
Epoch 19/60
46/46 [==============================] - 1s 17ms/step - loss: 0.5222 - accuracy: 0.8
035 - val_loss: 0.7739 - val_accuracy: 0.7517
Epoch 20/60
46/46 [==============================] - 1s 17ms/step - loss: 0.5138 - accuracy: 0.8
087 - val_loss: 0.7333 - val_accuracy: 0.7326
Epoch 21/60
46/46 [==============================] - 1s 17ms/step - loss: 0.5004 - accuracy: 0.8
114 - val_loss: 0.7403 - val_accuracy: 0.7367
Epoch 22/60
46/46 [==============================] - 1s 17ms/step - loss: 0.4789 - accuracy: 0.8
186 - val_loss: 0.8192 - val_accuracy: 0.7231
Epoch 23/60
46/46 [==============================] - 1s 17ms/step - loss: 0.4708 - accuracy: 0.8
247 - val_loss: 0.8391 - val_accuracy: 0.7353
Epoch 24/60
46/46 [==============================] - 1s 17ms/step - loss: 0.4512 - accuracy: 0.8
336 - val_loss: 0.7910 - val_accuracy: 0.7285
Epoch 25/60
46/46 [==============================] - 1s 17ms/step - loss: 0.4365 - accuracy: 0.8
366 - val_loss: 0.7867 - val_accuracy: 0.7435
Epoch 26/60
46/46 [==============================] - 1s 17ms/step - loss: 0.4181 - accuracy: 0.8
496 - val_loss: 0.7752 - val_accuracy: 0.7326
Epoch 27/60
46/46 [==============================] - 1s 17ms/step - loss: 0.3930 - accuracy: 0.8
```

```
554 - val_loss: 0.7926 - val_accuracy: 0.7340
Epoch 28/60
46/46 [==============================] - 1s 17ms/step - loss: 0.3945 - accuracy: 0.8
591 - val_loss: 0.7723 - val_accuracy: 0.7462
Epoch 29/60
46/46 [==============================] - 1s 17ms/step - loss: 0.3932 - accuracy: 0.8
540 - val_loss: 0.8463 - val_accuracy: 0.7353
Epoch 30/60
46/46 [==============================] - 1s 17ms/step - loss: 0.3770 - accuracy: 0.8
550 - val_loss: 0.8277 - val_accuracy: 0.7381
Epoch 31/60
46/46 [==============================] - 1s 16ms/step - loss: 0.3362 - accuracy: 0.8
806 - val_loss: 0.8109 - val_accuracy: 0.7449
Epoch 32/60
46/46 [==============================] - 1s 17ms/step - loss: 0.3552 - accuracy: 0.8
704 - val_loss: 0.7971 - val_accuracy: 0.7503
Epoch 33/60
46/46 [==============================] - 1s 17ms/step - loss: 0.3276 - accuracy: 0.8
840 - val_loss: 0.8420 - val_accuracy: 0.7340
Epoch 34/60
46/46 [==============================] - 1s 17ms/step - loss: 0.2936 - accuracy: 0.8
932 - val_loss: 0.8728 - val_accuracy: 0.7449
Epoch 35/60
46/46 [==============================] - 1s 17ms/step - loss: 0.2930 - accuracy: 0.8
898 - val_loss: 0.8690 - val_accuracy: 0.7476
Epoch 36/60
46/46 [==============================] - 1s 17ms/step - loss: 0.3136 - accuracy: 0.8
844 - val_loss: 0.8333 - val_accuracy: 0.7462
Epoch 37/60
46/46 [==============================] - 1s 17ms/step - loss: 0.2762 - accuracy: 0.9
059 - val_loss: 0.8426 - val_accuracy: 0.7640
Epoch 38/60
46/46 [==============================] - 1s 17ms/step - loss: 0.2612 - accuracy: 0.9
048 - val_loss: 0.9501 - val_accuracy: 0.7394
Epoch 39/60
46/46 [==============================] - 1s 17ms/step - loss: 0.2708 - accuracy: 0.9
052 - val_loss: 1.0670 - val_accuracy: 0.7367
Epoch 40/60
46/46 [==============================] - 1s 17ms/step - loss: 0.2686 - accuracy: 0.9
072 - val_loss: 1.1628 - val_accuracy: 0.7299
Epoch 41/60
46/46 [==============================] - 1s 17ms/step - loss: 0.3146 - accuracy: 0.8
885 - val_loss: 0.9346 - val_accuracy: 0.7271
Epoch 42/60
46/46 [==============================] - 1s 17ms/step - loss: 0.2659 - accuracy: 0.9
025 - val_loss: 0.8501 - val_accuracy: 0.7667
Epoch 43/60
46/46 [==============================] - 1s 17ms/step - loss: 0.2212 - accuracy: 0.9
192 - val_loss: 0.9953 - val_accuracy: 0.7353
Epoch 44/60
46/46 [==============================] - 1s 17ms/step - loss: 0.2226 - accuracy: 0.9
154 - val_loss: 0.9358 - val_accuracy: 0.7667
Epoch 45/60
46/46 [==============================] - 1s 17ms/step - loss: 0.2208 - accuracy: 0.9
188 - val_loss: 0.9009 - val_accuracy: 0.7544
Epoch 46/60
```

```
46/46 [==============================] - 1s 17ms/step - loss: 0.2169 - accuracy: 0.9
253 - val_loss: 0.9819 - val_accuracy: 0.7490
Epoch 47/60
46/46 [==============================] - 1s 17ms/step - loss: 0.1910 - accuracy: 0.9
294 - val_loss: 0.9992 - val_accuracy: 0.7572
Epoch 48/60
46/46 [==============================] - 1s 17ms/step - loss: 0.2285 - accuracy: 0.9
233 - val_loss: 1.0239 - val_accuracy: 0.7531
Epoch 49/60
46/46 [==============================] - 1s 17ms/step - loss: 0.1873 - accuracy: 0.9
345 - val_loss: 1.1042 - val_accuracy: 0.7572
Epoch 50/60
46/46 [==============================] - 1s 17ms/step - loss: 0.2028 - accuracy: 0.9
321 - val_loss: 0.9734 - val_accuracy: 0.7585
Epoch 51/60
46/46 [==============================] - 1s 17ms/step - loss: 0.1668 - accuracy: 0.9
393 - val_loss: 1.1072 - val_accuracy: 0.7531
Epoch 52/60
46/46 [==============================] - 1s 17ms/step - loss: 0.1936 - accuracy: 0.9
294 - val_loss: 1.0176 - val_accuracy: 0.7599
Epoch 53/60
46/46 [==============================] - 1s 17ms/step - loss: 0.1500 - accuracy: 0.9
495 - val_loss: 1.0397 - val_accuracy: 0.7599
Epoch 54/60
46/46 [==============================] - 1s 17ms/step - loss: 0.1534 - accuracy: 0.9
478 - val_loss: 1.1137 - val_accuracy: 0.7490
Epoch 55/60
46/46 [==============================] - 1s 17ms/step - loss: 0.1472 - accuracy: 0.9
492 - val_loss: 1.1271 - val_accuracy: 0.7572
Epoch 56/60
46/46 [==============================] - 1s 17ms/step - loss: 0.1416 - accuracy: 0.9
492 - val_loss: 1.2131 - val_accuracy: 0.7435
Epoch 57/60
46/46 [==============================] - 1s 17ms/step - loss: 0.1529 - accuracy: 0.9
482 - val_loss: 1.2705 - val_accuracy: 0.7326
Epoch 58/60
46/46 [==============================] - 1s 17ms/step - loss: 0.1631 - accuracy: 0.9
389 - val_loss: 1.2168 - val_accuracy: 0.7572
Epoch 59/60
46/46 [==============================] - 1s 17ms/step - loss: 0.1353 - accuracy: 0.9
505 - val_loss: 1.2320 - val_accuracy: 0.7544
Epoch 60/60
46/46 [==============================] - 1s 17ms/step - loss: 0.1257 - accuracy: 0.9
557 - val_loss: 1.2702 - val_accuracy: 0.7490
```

In [17]:
```
sns.lineplot(data=results_data, dashes=False).set(title='Training vs Validation Acc
```

Out[17]:
```
[Text(0.5, 1.0, 'Training vs Validation Accuracy')]
```



In [20]:

```
if (folder_url == test_examples["family_tiny_pad"]):
    landon = r"test_family_photos\tiny_padded_test\tiny-pad_landon_test_1.jpg"
    mike = r"test_family_photos\tiny_padded_test\tiny-pad_mike_test_1.jpg"
    robin = r"test_family_photos\tiny_padded_test\tiny-pad_robin_test_1.jpg"

    test_images = {"landon": landon, "mike": mike, "robin": robin}

elif (folder_url == test_examples["flowers"]):
    daisy = r"test_flower_photos\daisy_test.jpg"
    dandelion = r"test_flower_photos\dandelion_test.jpg"
    rose = r"test_flower_photos\rose_test.jpg"
    sunflower = r"test_flower_photos\sunflower_test.jpg"
    tulip = r"test_flower_photos\tulip_test.jpg"

    test_images = {"daisy": daisy, "dandelion": dandelion, "rose": rose, "sunflower

elif (folder_url == test_examples["family_no_pad"]):
    landon = r"test_family_photos\no_padding_test\no-pad_landon_test_1.jpg"
    mike = r"test_family_photos\no_padding_test\no-pad_mike_test_1.jpg"
    robin = r"test_family_photos\no_padding_test\no-pad_robin_test_1.jpg"

    test_images = {"landon": landon, "mike": mike, "robin": robin}

elif (folder_url == test_examples["family_padded"]):
    landon = r"test_family_photos\large_padded_test\padded_landon_test_1.jpg"
    mike = r"test_family_photos\large_padded_test\padded_mike_test_1.jpg"
    robin = r"test_family_photos\large_padded_test\padded_robin_test_1.jpg"

    test_images = {"landon": landon, "mike": mike, "robin": robin}
```

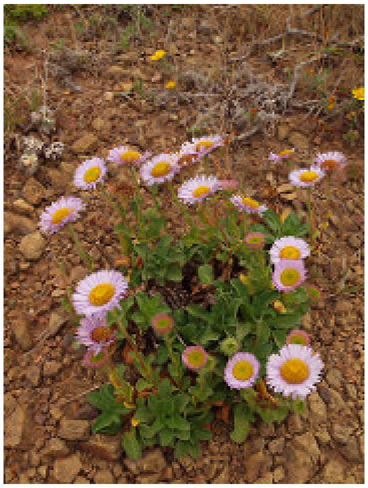
```
for test_image in test_images:
    print("Testing with " + test_image)

    final_test_img = tf.keras.utils.load_img(test_images[test_image], target_size=(
    img_array = tf.keras.utils.img_to_array(final_test_img)
    img_array = tf.expand_dims(img_array, 0) # Create a batch

    predictions = model.predict(img_array)
    score = tf.nn.relu(predictions[0])

    print(
        "This image most likely belongs to {} with a {:.2f} percent confidence. \n"
        .format(class_names[np.argmax(score)], 100 * np.max(score))
    )

    img = None
    img_array = None
    predictions = None
    score = None
```

```
Testing with daisy
This image most likely belongs to daisy with a 1383.15 percent confidence.

Testing with dandelion
This image most likely belongs to dandelion with a 649.88 percent confidence.

Testing with rose
This image most likely belongs to roses with a 1333.58 percent confidence.

Testing with sunflower
This image most likely belongs to sunflowers with a 793.11 percent confidence.

Testing with tulip
This image most likely belongs to tulips with a 1468.01 percent confidence.
```

In [19]:

```
for test_image in test_images:
    print("test photo: " + test_image)
    display(PIL.Image.open(test_images[test_image]))
    print()
```

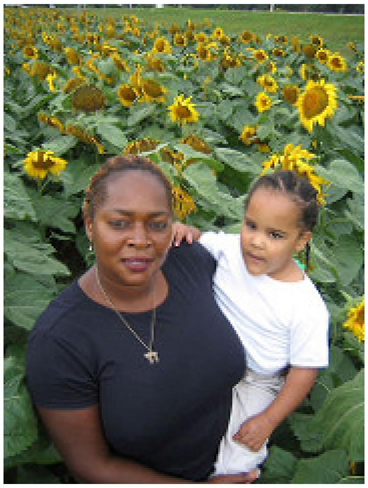
```
test photo: daisy
```



test photo: dandelion

test photo: rose

test photo: sunflower



test photo: tulip