A

Overview

A

A A A A A A fA c A A c A n A A G A A A
c A n A A A c A A A A n A A A A fA A A A
w A A A A c A A A A n A A n A A A A A
A A c A A fA A A A r n A A

- A n A A A c A A A A A A n c A fA A A A
- A A A fA A w A A A c A fb A A A y A n A A A n A
  fb A A A A A A A n A c w A A
- A A c A A n A A A n A A c A A A A A
  A n A A n A A A c A fb n A A A A A A A
  A A A A fb A A c A n A A A n A fb A A
- A A y A fA A n A A A c A A A A A A A w A A
  A c A A fb A A b n A A n A A A A c A
- n n A A b y c A A n A A A A w A fA A A A fb A A A fA
  A A A c A A A

A

A A b n A A A n A A w A fb A A A A A A A A A A
n A c A A A A A A A A A A A A A A A w A A
A A c A A A A n n A b A n A A A A fb A A b A n A A w A

A

A

Pre-requisites.

A

A A n A A w A c A y A A n A c A A A A w A b A A
A A A A A y n A A c A A A w n A A

A

n A fA A c A A A n A A

- y A A n y A
- W c A n w c A
- fA A A n A
- G A A n fb A
- A A n A
- A A n A
- A n A
- A A n A

A

Running Actions in Parallel

A A

A A G A A A n A A n A n A A n A A A A A A A A A A
fA n A A A n y n A A n A A A fb A w A A A A A A A n A A A

Program1.py

`on_for_seconds()`

`block`

`block=False`

Program2.py

Program3.py

Program4.py

A
A
A A A A A A A A A A A A A A A n A y n A A A
A A A A A A fb A A A n A A A A A c A A fb A A
fb A A n A A
A
A

Program5.py

A
A

fb A A A n A A A A A A A A A An A A A
A A A A **do not** A AA Ac A A A A A **exactly once** A A
Afb A Ac A Ac fb A A A A A A wait_until_not_moving() Ac AA
A A w A A A A Ac A A A A A A A A A A
before A A An A A AA A A AA A A A A An n A A
c fb A A n A A A A A An n An A
A
A A Ac A A A
A
AA

A better solution: Threads

AA
A A A
A
A A AA n A A A A Aw A A A w A fb A r A A A A
n AAfb A n A A A y A n A A A A A A A A A A A A
A AA A A A A A A A A A A AC A An A
n A A A A A A A A A A A A n A
A A An n A A A Ac A A A
A
fb A A A An A A An A n A A A A A fb A A A
A A A A A A n A A A A A A A A Afb A AA A
w n A A A A Ac A A A A A A Afb A
A
fb A A w A A n A *Program2.py* A
A

A
A
A A A A A A A Ac A A AA onForSeconds() Afb A A
A A n AA motorA speedA Aseconds AA A A A A A n A A
A fb A A A A A Afb A A A A A An A AA n n Afb AA A Afb A
A Ac n A A A A A An n A A A A A
A
A

Program6.py

A

A

A n Ac A A A A *Python6.py* A

A

A

Python7.py

A
A
A A A A A A fbA A An Ac A AAfb A A
onForSeconds()A A A A r A n AA A A A Afb A A A
n An n A A A A A blockTrueA n A A A A A A An A A
A
A

A
A
A n AonForSeconds()Afb A Ac A A AA A A A A A A
c AA Afb A A AA A AA AtargetA n A fb A Afb A A A
A A A A A A A A AonForSecondsA A AargsA n AAA Afb
n A A A A Afb A
A

A A AA fb A A A AAnn Ac A A A start() An A A A A
A A A

A

A

[redacted]

A

A

A AA A A A args A n AA A A AfbAw A A An Ac A A A n A
A A A n A A A A Afb Ac A AA AfbA Afb A A
A A n A An A fbA AA A n n A fb A Aw Ac A A A A

args = (30,) AAA

A

An An A A A AAA An AfbA A A A n A A A An A
w A A A A n AA A A n n A fb A Aw A A AA A n A
n A A A A A A A A c n AAA

A

A A AA AAA A A A AA w A A threadPool A A A A A
A Ar A A A A AfbA A A A A A n AA A A A A A A
Python7.py, A A An A A A A nA A A threadPool A Ac fb A A
n A

A

A

[redacted]

A

A

A A threadPool A A A A A A AfbA A A A A A A AfbA
A A A w A A A isAlive() An AAfbA A A A n A A n w A Afb n A A
A A A n A A A A A A AAn A

A

A AfbA A while threadPool: A A A AfbA A A A n A A AA A A A A
A A An A AAA A A A n A A nA A A A A y A
A

A

A

[redacted]

[redacted]

A

A

An A wA A A A A A A A A Afb A wA A A A A
A Afb A nAc wA A A Acw A A Ac A A Afb A A A Ac A
A A A An w A A Afb A n A `waitUntilAllThreadsComplete()`.A

A

A

[redacted]

Python8.py

AA
A
w A A A A A Ay A A AAfb A A A AA A w A fa
A A A Afb A A A A A A An A AAn A
A A A A A A A nA Ay A A A A A r A AA n yA A
A
Am Ac AAc A A fa c A A A A A A c A A fb n A A w A
A A A n Ac A fa A AA A nAc A A A A Actions A
A A A A A A A A c A A fb n AA A A A fb n A A A A
A A A fb A Afb Ac AA A A A A Actions A A A
fb n A A A Ac A A
A
AcreateAction()Afb A A A n A A A Ac A A A
onForSeconds()Afb A A A nA AA n A AAfa A A fbn A A
A Am A A A fb A A A A A A **Pre-requisites** A A A A A fa A
AA
A
n A Actions A A A A A fa AA
A
AA

```
#!/usr/bin/env python3

from ev3dev2.motor import MediumMotor, LargeMotor, OUTPUT_B, OUTPUT_C
import threading

def onForSeconds(motor, speed, seconds):
    motor.on_for_seconds(speed, seconds, brake = True, block = True)

def createAction(name, motor, speed, seconds):

    action = {}
    action['name'] = name
    action['motor'] = motor
    action['speed'] = speed
    action['seconds'] = seconds

    return action
```

```
def main():

    actions = []

    largeMotor_Left = LargeMotor(OUTPUT_B)
    largeMotor_Right = LargeMotor(OUTPUT_C)
    mediumMotor = MediumMotor()

    action1 = createAction('onForSeconds', largeMotor_Left, 20, 4)
    action2 = createAction('onForSeconds', largeMotor_Right, 40, 3)
    action3 = createAction('onForSeconds', mediumMotor, 10, 8)

    actions.append(action1)
    actions.append(action2)
    actions.append(action3)

    for action in actions:
        if action.get('name') == 'onForSeconds':
            onForSeconds(action.get('motor'), action.get('speed'), action.get('seconds')

main()
```

Python9.py

A

```
element 1, element 2, element 3,
```

C A A A A fb A n A A A A A A A A

```python
#!/usr/bin/env python3

innerArray1 = [1, 2, 3]
innerArray2 = [4, 5, 6]
outerArray = [innerArray1, innerArray2 7, 8]

for element in outerArray:
  print("outer {}, ".format(element), end = "")

print("")
```

C A A A A A A A A A A A A A A fb A n A fb A
outerArray A A A Aw A A A A A A A A A A A A
A fb n A A A A A Ar Ac A A **has not** A A A
A A fb A A A

A

```
outer [1, 2, 3], outer [4, 5, 6], outer 7, outer 8,
```

A A A A A A A A fc A A A A A A

A fb A A A A A outerArray A A A A w A y n AG A A n A
fb A A A A A A n A A A A if isinstance(element, list): A
A A A n A A A A A A w A A A A A A A
A A A A A A A c n A fb A A n A A A A A A A
n A A A Aw A

```python
#!/usr/bin/env python3

innerArray1 = [1, 2, 3]
innerArray2 = [4, 5, 6]
outerArray = [innerArray1, innerArray2 7, 8]

for element in outerArray:
```

```
  if isinstance(element, list):

    print("*", end = "")
    for subElement in element:
        print("inner {}, ".format(subElement), end = "")
    print("*")

  else:
    print("outer {}, ".format(element), end = "")

print("")
```

A
A
A n A c w A A A A b A A A A A w A A w A
fb n A A A n A A w A A w A b n A A A A A A A
fb A w A w A b n A A A A A A c A fy A n A A A A
A A A w A
A
A

```
*inner 1, inner 2, inner 3,*
*inner 4, inner 5, inner 6,*
outer 7, outer 8,
A
```

A
A
A A A A A A
A
A A
A fa A A A
A

```
...

def main():

    actions = []

    largeMotor_Left = LargeMotor(OUTPUT_B)
    largeMotor_Right = LargeMotor(OUTPUT_C)
    mediumMotor = MediumMotor()

    action1 = createAction("onForSeconds", largeMotor_Left, 20, 4)
    action2 = createAction("onForSeconds", largeMotor_Right, 40, 3)
    action3 = createAction("onForSeconds", mediumMotor, 10, 8)

    actionParallel = []
    actionParallel.append(action1)
    actionParallel.append(action2)
```

```
    actions.append(actionParallel)
    actions.append(action3)

    for
```

```
    action['speed'] = speed
    action['seconds'] = seconds

    return action

def main():

    threadPool = []
    actions = []

    largeMotor_Left = LargeMotor(OUTPUT_B)
    largeMotor_Right = LargeMotor(OUTPUT_C)
    mediumMotor = MediumMotor()

    action1 = createAction("onForSeconds", largeMotor_Left, 20, 4)
    action2 = createAction("onForSeconds", largeMotor_Right, 40, 3)
    action3 = createAction("onForSeconds", mediumMotor, 10, 8)

    actionParallel = []
    actionParallel.append(action1)
    actionParallel.append(action2)

    actions.append(actionParallel)
    actions.append(action3)

    for action in actions:

        # are their multiple actions to execute in parallel?
        if isinstance(action, list):

            for subAction in action:
                if subAction.get('name') == "onForSeconds":
                    thread = threading.Thread(target = onForSeconds, args =
(subAction.get('motor'), subAction.get('speed'), subAction.get('seconds'))
                    threadPool.append(thread)
                    thread.start()

        # is there a single action to execute?
        else:

            if action.get('name') == "onForSeconds":
                thread = threading.Thread(target = onForSeconds, args =
(action.get('motor'), action.get('speed'), action.get('seconds'))
                threadPool.append(thread)
                thread.start()

        waitUntilAllThreadsComplete(threadPool)
```

```
main()
```

Python11.py

A
A A A A A A A y A b A A A
A
A

```
...

def onForSeconds(motor, speed, seconds):
    motor.on_for_seconds(speed, seconds, brake = True, block = True)

def delayForSeconds(seconds):
    sleep(seconds)

def createAction(name, motor, speed, seconds):

    action = {}
    action['name'] = name
    action['motor'] = motor
    action['speed'] = speed
    action['seconds'] = seconds

    return action

def launchStep(action):

    if action.get('name') == "onForSeconds":
        thread = threading.Thread(target = onForSeconds, args = (action.get('motor'),
action.get('speed'), action.get('seconds')))
        thread.start()
        return thread

    if action.get('name') == "delayForSeconds":
        thread = threading.Thread(target = delayForSeconds, args = (action.get('seconds'),
))
        thread.start()
        return thread

def main():

    threadPool = []
    actions = []

    largeMotor_Left = LargeMotor(OUTPUT_B)
    largeMotor_Right = LargeMotor(OUTPUT_C)
    mediumMotor = MediumMotor()

    action1 = createAction("onForSeconds", largeMotor_Left, 20, 4)
```

```
    action2 = createAction("onForSeconds", largeMotor_Right, 40, 3)
    action3 = createAction("delayForSeconds", None, None, 2)
    action4 = createAction("onForSeconds", mediumMotor, 10, 8)

    ...
```

Python12.py

Stopping the Threads

Stopping the Threads (Eventually)

```
#!/usr/bin/env python3

from ev3dev2.motor import MediumMotor, LargeMotor, OUTPUT_B, OUTPUT_C
from ev3dev2.sensor.lego import TouchSensor
from time import sleep

import threading

def onForSeconds(motor, speed, seconds):
    motor.on_for_seconds(speed, seconds, brake = True, block = True)

def delayForSeconds(seconds):
    sleep(seconds)

def createAction(name, motor, speed, seconds):

    action = {}
    action['name'] = name
    action['motor'] = motor
    action['speed'] = speed
```

```
    action['seconds'] = seconds

    return action

def launchStep(action):

    if action.get('name') == "onForSeconds":
        thread = threading.Thread(target = onForSeconds, args = (action.get('motor'),
action.get('speed'), action.get('seconds')))
        thread.start()
        return thread

    if action.get('name') == "delayForSeconds":
        thread = threading.Thread(target = delayForSeconds, args = (action.get('seconds'),
))
        thread.start()
        return thread

def main():

    threadPool = []
    actions = []
    stopProcessing = False

    largeMotor_Left = LargeMotor(OUTPUT_B)
    largeMotor_Right = LargeMotor(OUTPUT_C)
    mediumMotor = MediumMotor()
    ts = TouchSensor()

    action1 = createAction("onForSeconds", largeMotor_Left, 20, 4)
    action2 = createAction("onForSeconds", largeMotor_Right, 40, 3)
    action3 = createAction("delayForSeconds", None, None, 2)
    action4 = createAction("onForSeconds", mediumMotor, 10, 8)

    actionParallel = []
    actionParallel.append(action1)
    actionParallel.append(action2)

    actions.append(actionParallel)
    actions.append(action3)
    actions.append(action4)

    for action in actions:

        # are their multiple actions to execute in parallel?
        if isinstance(action, list):

            for subAction in action:
```

```
                thread = launchStep(subAction)
                threadPool.append(thread)

        # is there a single action to execute?
        else:

            thread = launchStep(action)
            threadPool.append(thread)

        while not stopProcessing:

            # remove any completed threads from the pool
            for thread in threadPool:
                if not thread.isAlive():
                    threadPool.remove(thread)

            # if there are no threads running, exist the 'while' loop
            # and start the next action from the list
            if not threadPool:
                break

            # if the touch sensor is pressed then complete everything
            if ts.is_pressed:
                stopProcessing = True

            sleep(0.25)

        # if the 'stopProcessing' flag has been set then break out of the program
altogether
        if stopProcessing:
            break


main()
```

Python13.py

A

A

A A n A c w A A A A A n A A A A A A A A n A A A A n n A c A A A f b A A f b A A A A n A A A A A A motor.on_for_seconds() A A A A f b A A f b A n A A f b A A A A c A A A A A A A A A A c A n n A A A W A w n A

A

A

Stopping the Threads (Immediately)

A

A A A A A A A n A A A A A c A A f b A A

A

A Ac A A A A A A c A A A A A AAfb A
n A A Ac A A A fb n A *Python13.py* A A A A fb A AA
A A A A fb A A A A c A A A fb A A n A A
A
A

```
#!/usr/bin/env python3

from ev3dev2.motor import MediumMotor, LargeMotor, OUTPUT_B, OUTPUT_C
from ev3dev2.sensor.lego import TouchSensor
from time import sleep

import threading
import time

def onForSeconds(stop, motor, speed, seconds):

    start_time = time.time()
    motor.on(speed, brake = True, block = False)

    while time.time() < start_time + seconds:

        # if we are stopping prematurely break out of loop
        if stop():
            break

    motor.off()

def delayForSeconds(stop, seconds):

    start_time = time.time()

    while time.time() < start_time + seconds:

        if stop():
            break

def createAction(name, motor, speed, seconds):

    action = {}
    action['name'] = name
    action['motor'] = motor
    action['speed'] = speed
    action['seconds'] = seconds

    return action

def launchStep(stop, action):

    if action.get('name') == 'onForSeconds':
```

```
        thread = threading.Thread(target = onForSeconds, args = (stop, action.get('motor'),
action.get('speed'), action.get('seconds')))
        thread.start()
        return thread

    if action.get('name') == 'delayForSeconds':
        thread = threading.Thread(target = delayForSeconds, args = (stop,
action.get('seconds')))
        thread.start()
        return thread

def main():

    threadPool = []
    actions = []
    stopProcessing = False

    largeMotor_Left = LargeMotor(OUTPUT_B)
    largeMotor_Right = LargeMotor(OUTPUT_C)
    mediumMotor = MediumMotor()
    ts = TouchSensor()

    action1 = createAction('onForSeconds', largeMotor_Left, 20, 4)
    action2 = createAction('onForSeconds', largeMotor_Right, 40, 3)
    action3 = createAction('delayForSeconds', None, None, 2)
    action4 = createAction('onForSeconds', mediumMotor, 10, 8)

    actionParallel = []
    actionParallel.append(action1)
    actionParallel.append(action2)

    actions.append(actionParallel)
    actions.append(action3)
    actions.append(action4)

    for action in actions:

        # are their multiple actions to execute in parallel?
        if isinstance(action, list):

            for subAction in action:
                thread = launchStep(lambda:stopProcessing, subAction)
                threadPool.append(thread)

        # is there a single action to execute?
        else:
            thread = launchStep(lambda:stopProcessing, action)
            threadPool.append(thread)
```

```
        while not stopProcessing:

            # remove any completed threads from the pool
            for thread in threadPool:
                if not thread.isAlive():
                    threadPool.remove(thread)

            # if there are no threads running, exist the 'while' loop
            # and start the next action from the list
            if not threadPool:
                break

            # if the touch sensor is pressed then complete everything
            if ts.is_pressed:
                stopProcessing = True

            sleep(0.25)

        # if the 'stopProcessing' flag has been set then break out of the program
altogether
        if stopProcessing:
            break


main()
```

Python14.py

A

A

A A A A A A A Aw A Afb A A fb n A A A A A

A A A A n A A A A A A A A A A A A

A A A n A A A An A A Afb A delayForSeconds() A

A

C A A A A AA A A motor.on_for_seconds() Afb A A sleep() A

n n A A A c A

A

A

A

A

A A A A Ac A A A A Ac A AA A A A A

A A A n A A w Afb n A A A n AA A A A n A A A

A A n A A A A A A A n A A A A A A A A n A

A A A

A

A A n A stop A A A A fb A A A A Ac A A Afb A
w A A True A A n Ac A A fb A while A A A A A n A A A
A
A
A

A
A
A onForSeconds() Afb A A Ac A A A A A Ac A A A A A
delayForSeconds() Afb A A A A A n Ac fb A A A An A
An A A A A A A A fb fb fb An An A Ay A A
A A n A A A A True A Afb A A A A A n A
A
A

A
A
A A An Ac A A A A fb A A A A A A A A A Ac A fb A
An A A A n A n A A A A A fb fb Ac A A A A A A A A
A A A nA A A A A A A A n A
A
A A A A A A A A A A n A A A Afb A A AA A
n Ac A A A A Afb A A A w A w An A A A fb A A A A A
A A An A A Ac A A Ac A A fb A A A A A delayForSeconds() A A
onForSeconds() Afb A
A
fb A A A A main() Afb A A A A A Afb A A A Ac A A A A
w c A stopProcessing A A A n AC A w A A A A A A A A A A
A A

A

A A A A A A A A A A A A A A Lambda: A A A

A A Ac A A A Afb A A AAw A A AA Ac Aw A A

w A A A A AA A A A A A y A A A A A

A A A AA A A AStopProcessingAw c A A Amain()Afb A A Ac A

w A A A A A A An A

A

A

A

A

G Afb A A A A AonForSeconds()Afb A A A A Aw A

A Aw c A AAA fb A A AAfb A A A Aw c A A AA

fb A AAfb A A Ac A w A

A

A

A

A

N A fb n A A n c A y A Ac Afb A AA

fb n n c n c y A

A

A

Specifying our Robot Actions in a File

A

A A A A wA A c A fb n A A A A A A A A A A A A

A A A A n A An wA Afb A A Afb AA A A A A A Afb n AA

n A y Afb A A A r Afb A fb A A A Ac A n A A r Ac A A A

fb y c A A A fbA A AA

A

A A A A A A A Afb A AYN A A AAn A A A A

n A n yA A A A A c A AA A A A A A n y Afb A r n A

An A

A

A

Reading from Text Files

Python15.py

A

Python16.py

A

A Afb n A A n A c w A A Ac A

A

A

A

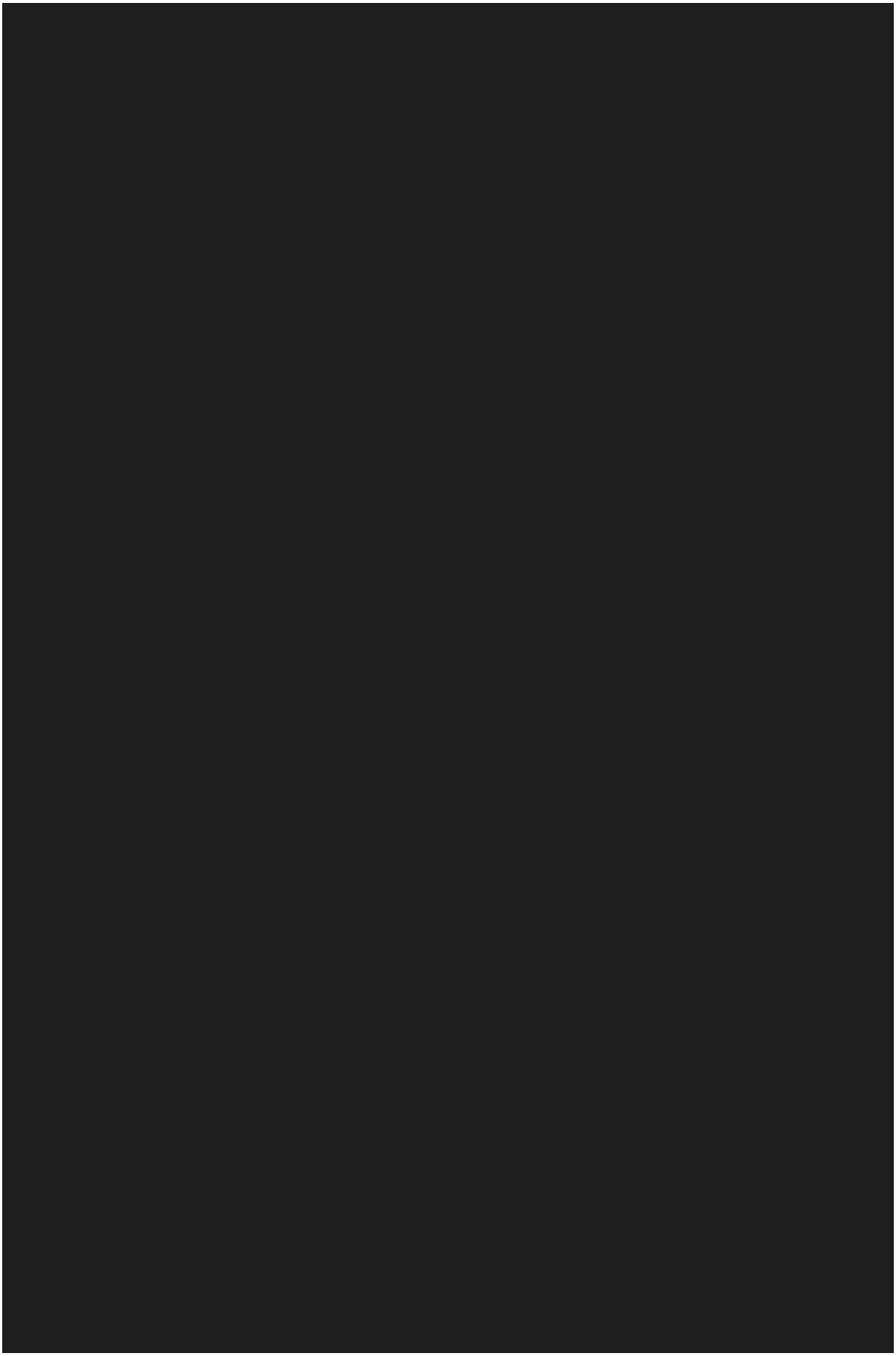
A

A A A A A A A A n n Afb n A A y Afb A A A A A A A

A AA A ffb A A fa A A Ac A

A

A



Python17.py

A

A

A A A w A A A Afb A A A Ac AA A A A Afb n A A y A
fb A A w A A A c A A fbA A n A Ac A y A A AA A A A A
A Afb A A A A A Afb A A A A A Ac A
fb n Ac A An Ac A Ac A A

A

Recursion

A

C Ac fb A A A A A A A A A fbA A AA Ac A A A A A A A
A A AA A n Ac A n A A fb A

A

A

Python18.py

A

A

A A n A A A A Ac A A A A A W A AA

A
A

A
A
A A A A A n A A A fb n A A A A A A c A A A A c A A A A A A
A
A
A A n A fb A A A A A printNumber() fb A A A w A fb A A
A printNumber() fb A A A w A A A A A calls itself again A A A
n c A A A A A A A y A A A printNumber() A A A A w A fb A
c fb A A fb A A A A A A A A n c w A r A A A
A
A A A A n A fb A A A A A w A fb A A n A A
n A A A c A A A w A A fb printNumber() A A A A A
n A A A A A c fb A n A A A c A A A w A A fb
printNumber() A A A A n A A A A A c fb A n A A A
A A A A A A n A A A n A n A
A
A A A A printNumber() A A c w A A A c A
A
A

A A
A
A A A A A A y A A w A A A c A A w A fb A A
A

A

XML

A

A A w A A y Afb A A A A A Ac A A A Afb A fb A A
Ac A AA A Afb n A A A A AYN A AAn Ac Aj cA fa A
A Aj A A A A A A

A

C A AA n AYN Afb A An A A n A A A w Ac A A A A AA
A A A A A A A c A A<step>A A A Aw A fa A A n An A
A A A A A A n w A AA A A n A A A A A A n Ac A A
A A A A A A A A A<step>A A</step> AA A A A A A A
A A A A w An A A A AA A A A fb n A A A A A A A
A A AAV AA w A A A A A A n Ac A A fb A A
Ac A A AYN A A A A Ac A A

A

AYN A A A A A A n A A A n Aw A A A A c A

A

A

A

A

A A A AYN A A A A Aw A A A A A A A A A
c A nA Ac A n fb A A Ac AA

A

A

A

A

A AYN A A A A A A A Ac A fb fb A A Aw c A w A
n A A A A A A cA A A A A A A A A AYN AA w A
AYN Afb A A A A A fb AA n c A fb fb A Afb A A A A A

- Afb A n c A n n AA A A c A A A A n y A A A c c A
A A A n n A A Afb A A n A A A A A AA Afb A A A
A A A A fb A A

A

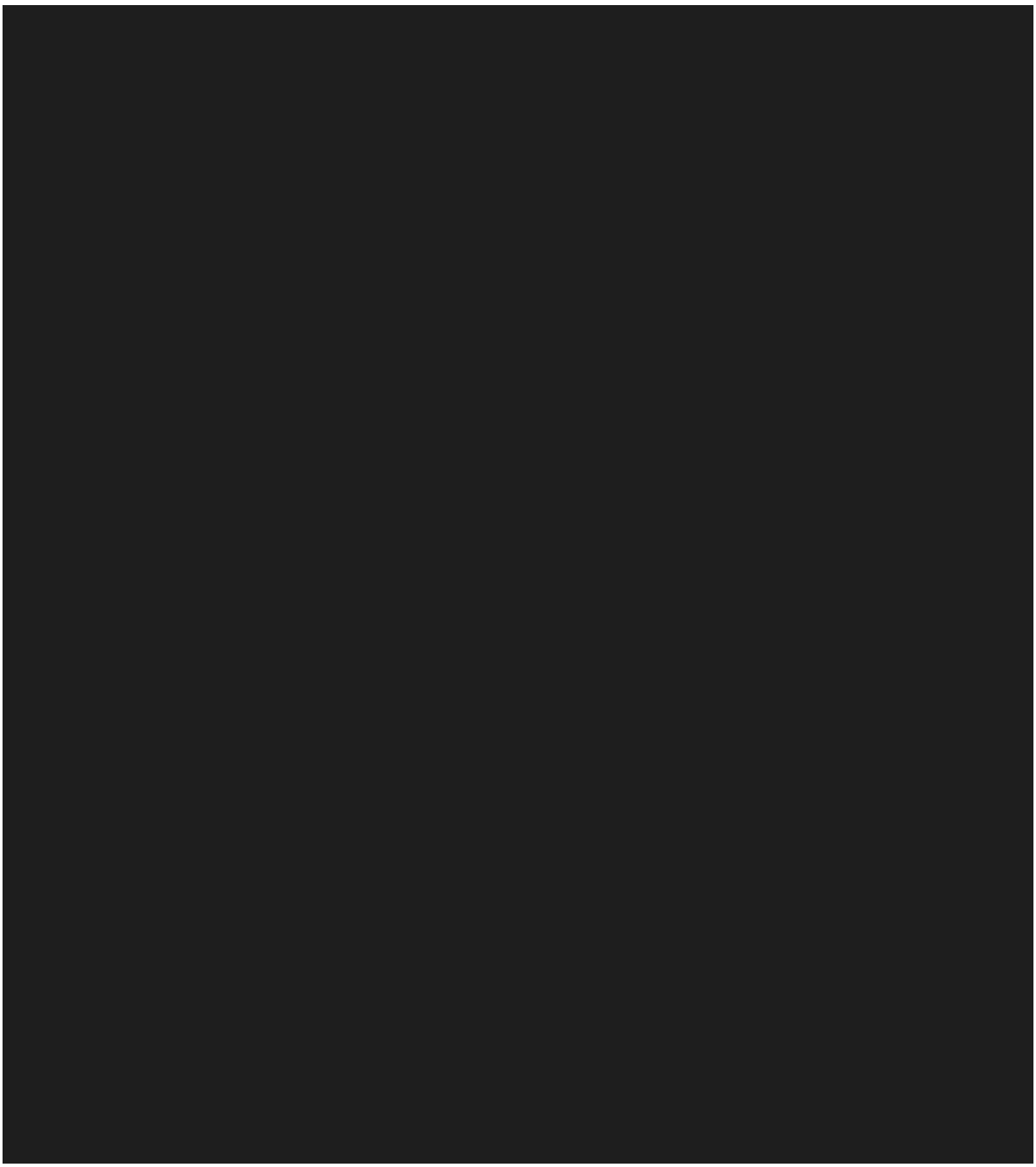
- YN A Ac Aw A A A AAA A A A AA wA A
  c A Aw A AA A Ac AA A Afb AY A n Aw A
  A w A A Aw A Aw A
  A
- An A AYN AA A A A A A A AA

A
n A n A Afb A A AYN Afb AA Ac AA A A A A Afb A A A A
A A AcJ A AxmlDocumentA Afb nA A A wA A w A A A
A A A A A A A Afb A A
A
A w A n A A A A nA A A A A Ac A A A
A A nn AforA n A A A wA Afb A A A
A
A

Python19.py

A
A
A Afa Ac wA nA A Ac A
A
A

A
A
C A A A A A n A A AYN A An A
A
A AA A A A A A n Afa A A n A A Afb A w A
A A AAfa A A A AA A A A A wA A w A A
A n AA A A A A n A A A
A
A nAc A A A Afa A n A A A n Afb AA An A A A A
loopThroughXML()Afb A n Afa A A n AAAlaunchInParallelA A
Afa An A A AloopThroughXML()fb A w A A A A AA
A
A

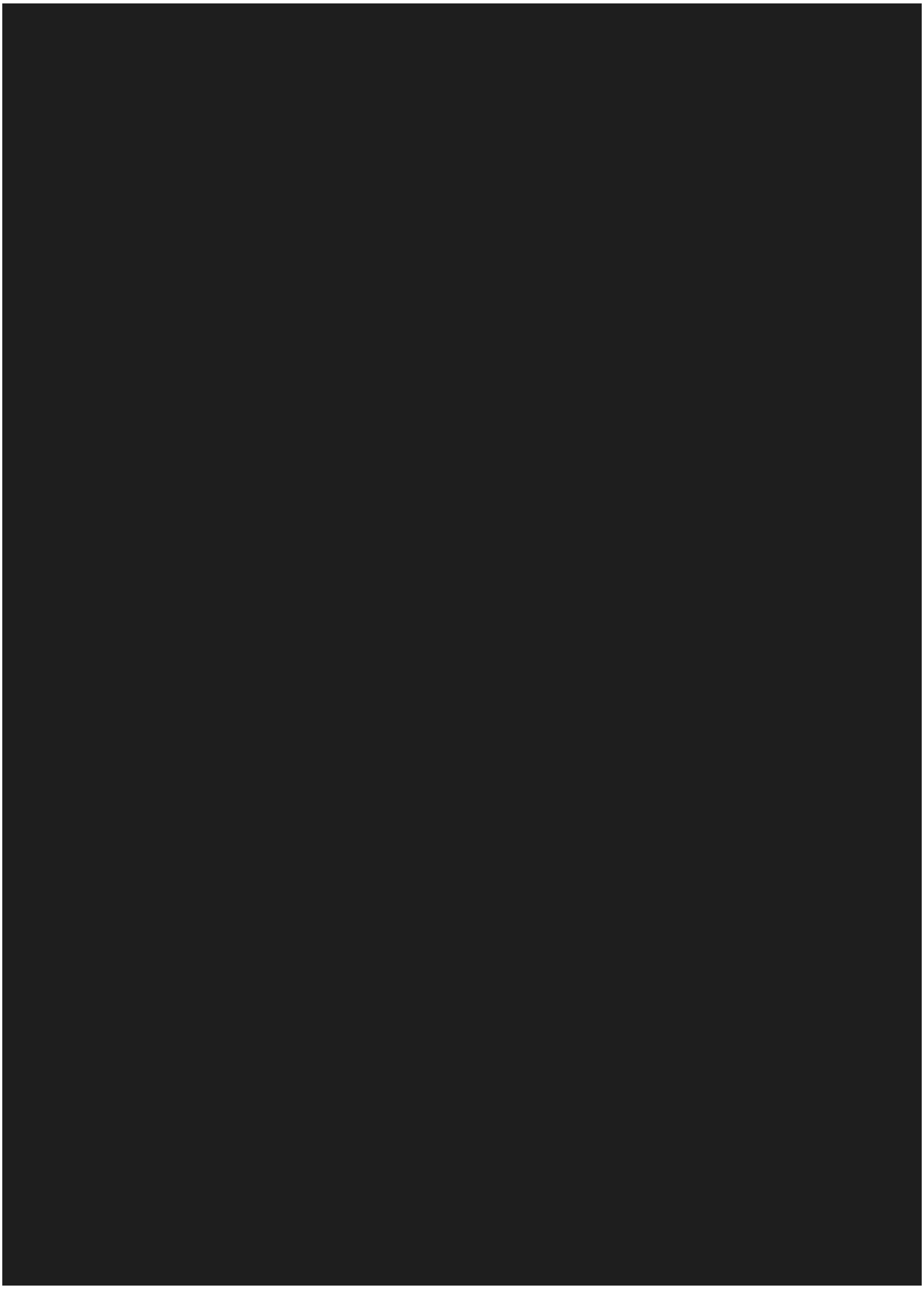


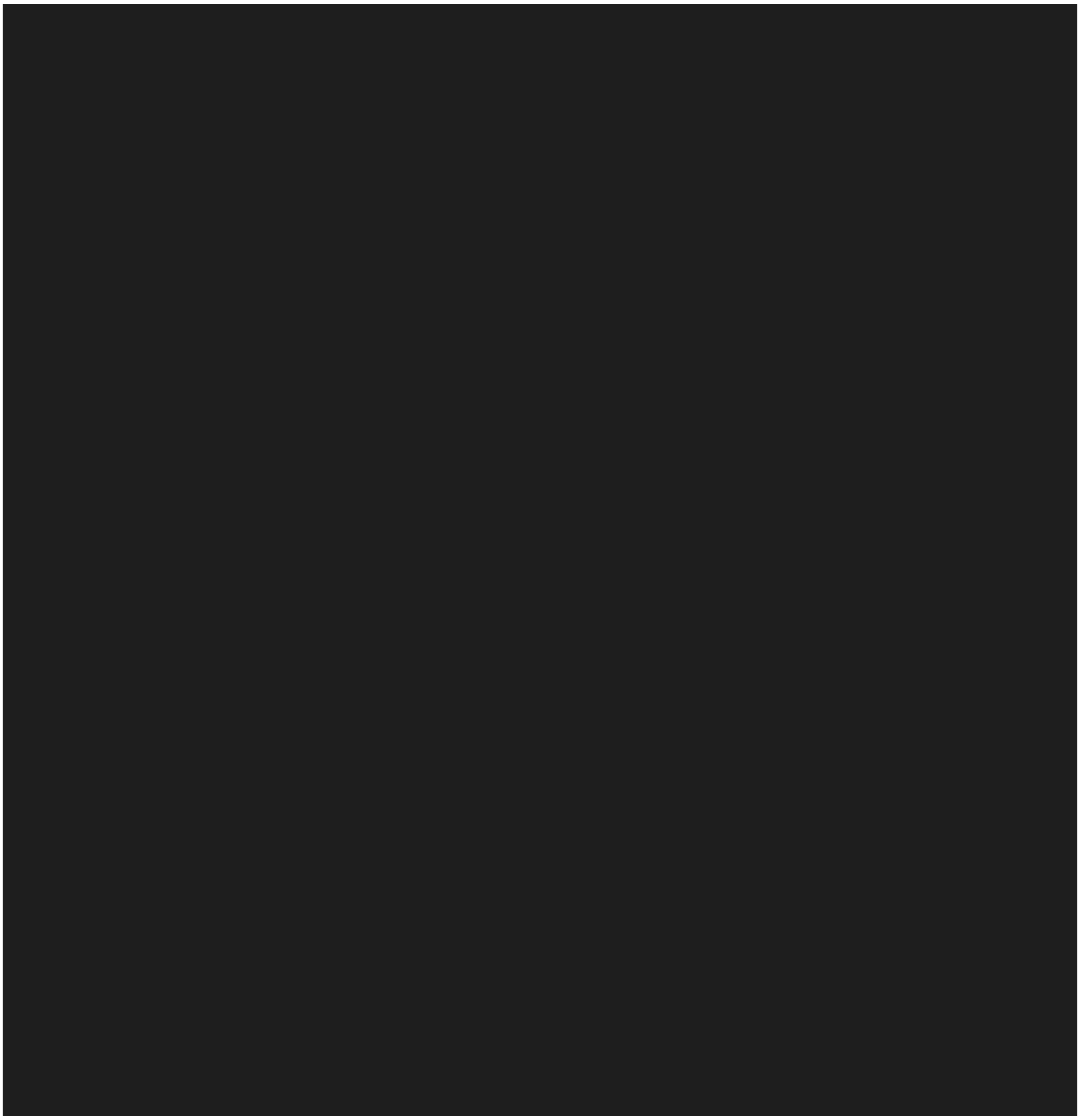
Python20.py

A
A
A y A A A n A A A A N A b A
A
A

A
A

A A A A A AYN Afb Afb A A A A A A Ac AA A A A A
flA A AA w A n w A A A A
A
A





Python21.py

A

A

A launchStep() Afb A A A A n A A A Afb A A A A

AA AYN A n A A action AA A A name A motor A speed A A seconds A

w c A A A Aw A w Afb n A AYN A n Ac fb A AA A A

A w A A

A

A A A A A speed A A seconds Aw c A A w A Afb A A n c A A

A Ay AAYN Afb A A A w AA A fbA A A A A Aw A Afb n A n A A

Aw AA A w AA A A n A A AA Aw A A An A A

Afb A y n A A A A A A

A

A

A

A

A A ﬂA AYN Aﬁb A A Aﬁb AA A A A A A n A n A

A A A w A n A A A A A A A A

launchStep() Aﬁb AA Ac ﬁb A A A A A A A A A A A A

n Aﬁb A n Ac ﬁb A AAAA

A

ﬂA A A n A A n A Ac A A launchInParallel A A A A n A A

A A A A A A n A

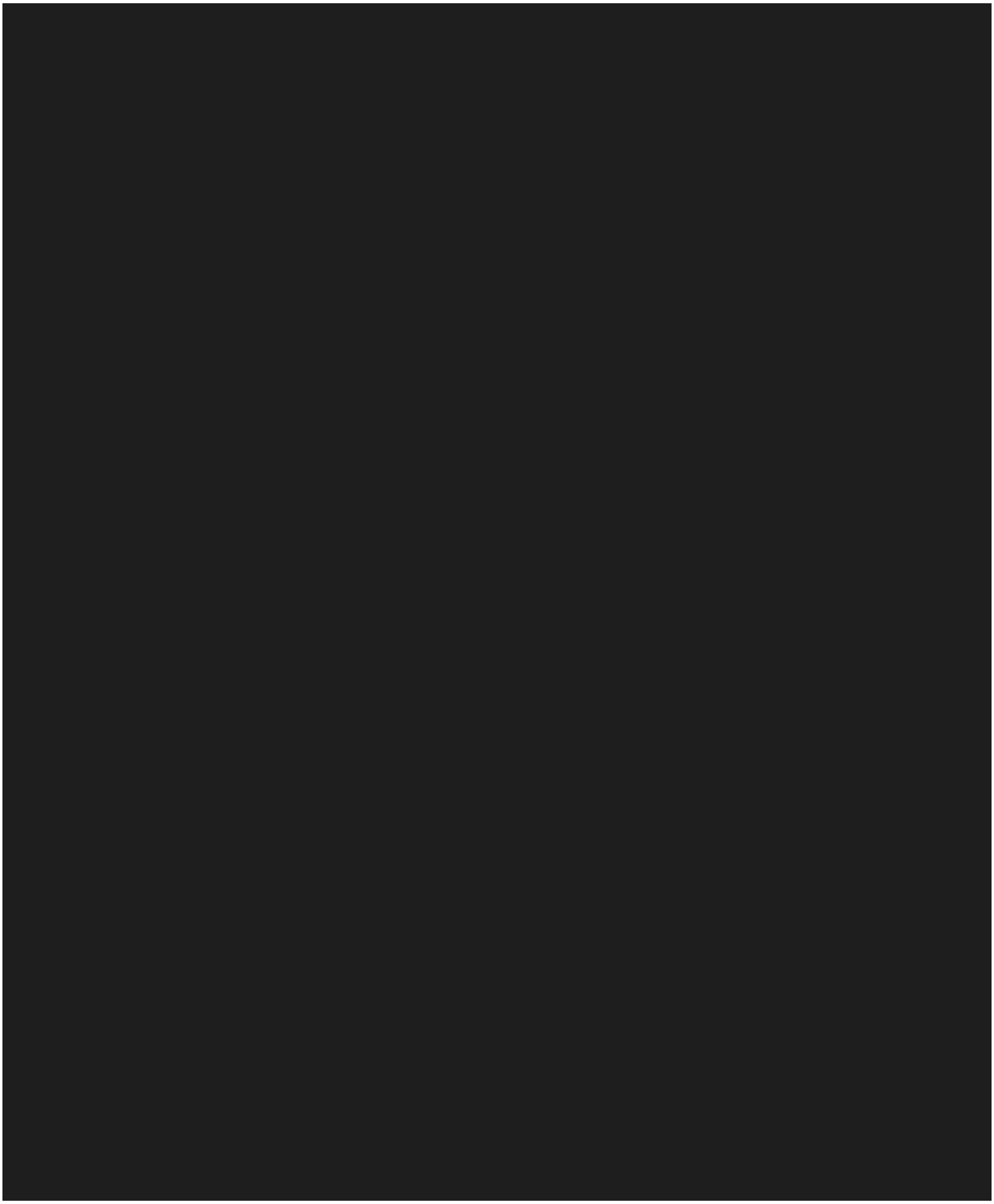
A

A

A

A

A A A A wA A A A A A A A A



Program22.py

A

A

A A fb A Ac A c c A w A A y AA A AAw Afb n A A
A A A An Ac A ffb Ac A A A A A fb AA A w n A A
A n Afb A A n A A A Ac A Ac A A y AAA

A

A

abs() … print(abs(-3)) … 3

```
if abs(rColourSensor - rProgram) < 20
if abs(104 - 100) < 20
if abs(4) < 20
if 4 < 20
```

abs() …

```
if abs(rColourSensor - rProgram) < 20
if abs(98 - 100) < 20
if abs(-2) < 20
if 2 < 20
```

abs() …

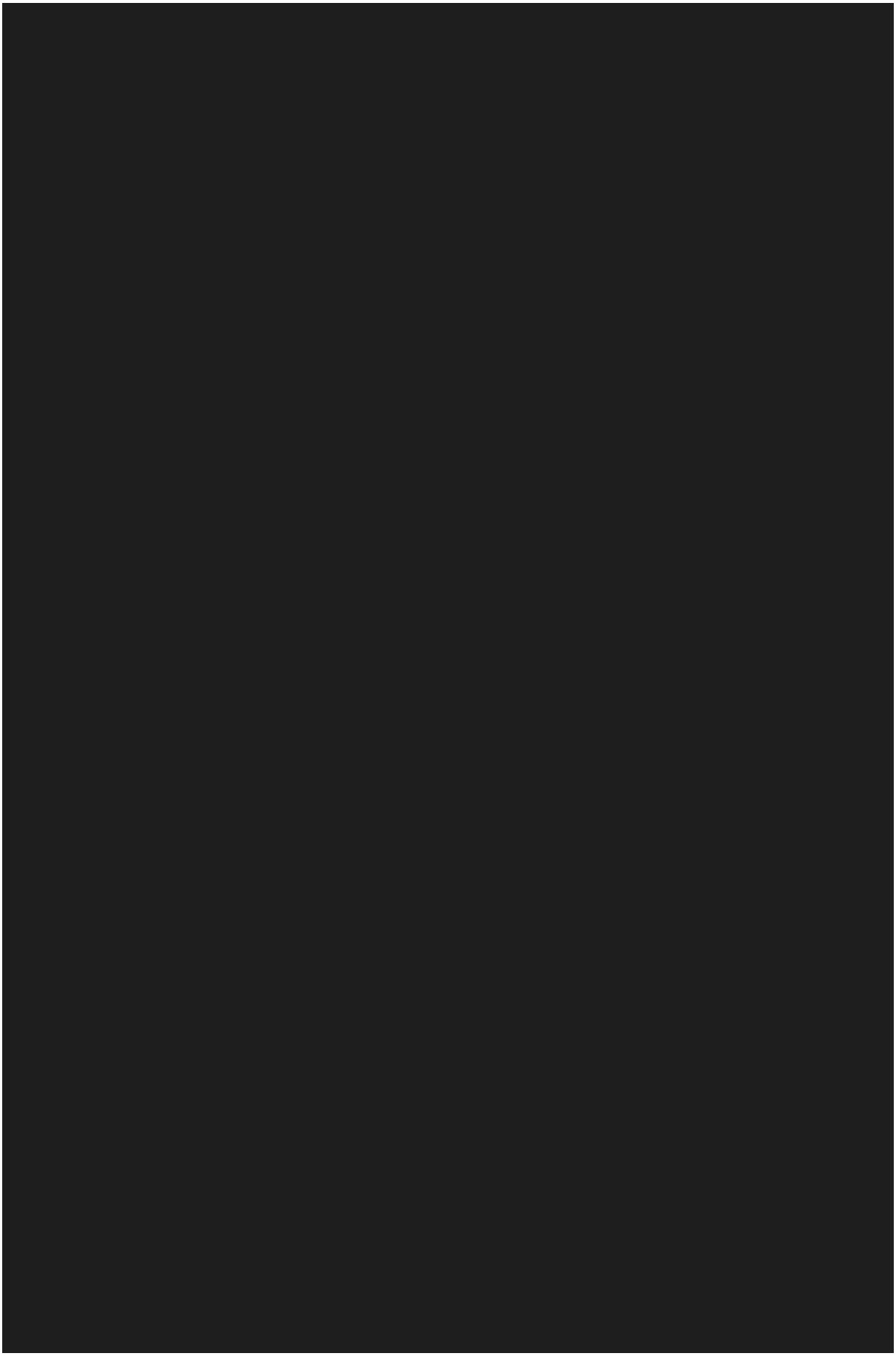
Run2 …

Ac AA A A Afb Afb A A A n A A AA A A w A A A
ffb A A A
A
A
[redacted]
A
A
A A A A A A Ac A A A A A A AA A A A
n AA A A A A A n Ac A
A
A
[redacted]

Python23.py

A
A
A n A c w A y A A A c A A A Y N A n A fb A A A A
A A A A n A Y N A A A A fb A A A n A fb A A A n A
A A A n A A A A
A
A A n A A A A c A A fb A fb A A A A A A A c A
A
A

A
A
A A Y N A A A A c A A A fb n A A A A n A fb n A fb A
c A A A n A c A n A A A A fb A n A A A A A A A
A main() A fb A A A A A A A A A A A A A A w A A A
A A A A n A
A
A



A
A
Afb A An A c A n Afb fb A Afb A fa A A cJ w A

- A w A A A A A
- A w A Ac A An n A
- A A c A A n A A A n A A Ac A AA A A A A
  fb A fa A A A A
- A A n A Ac A fb A A Ay Afb A A A A A

A A cJ w A Ac A n A Afb A A c A A A A A c A Ac A fb A fa A
c A A A A A A fa A A AA
A
A

Stopping the Robot when Lifted

A
Afb A A w A A A A A A A A fa A n AAfa A A A A A A
w An A A A A A A Afb A A A n A A A A
A A A w A AA A A A A Ac A A A A A A A An A Ac A
A A A A A c A Ac A fb A
A

Python25.py

A
A
Afb A A A A A w A A A AW A A A A A A A A A A w A
c A A A AA A A A A A w A A A A A A A c An A A A A
A A fbA A A A A fb n A A A
A
A Ac w A AA w A A A Aw c A ACOLOUR_SENSOR_MATA A
COLOUR_SENSOR_ATTACHMENTSA A A A A A A A A A AA
fb A A A A AA A A A Aw c A A Ac A
A
A

A
A

launchInParallel launchInSeries

- launchInParallel launchInParallel launchInSeries

Debugging a Program

Logging .. argghh!

print()

```
#!/usr/bin/env python3

print('EV3 Python')
print('EV3 {}'.format('Python'))
print('{} {}'.format('EV3', 'Python'))

firstWord = 'EV3'
secondWord = 'Python'
print('{} {}'.format(firstWord, secondWord))
```

print() EV3 Python

end file end

A

```
firstWord = 'EV3'
secondWord = 'Python'

print('{} '.format(firstWord), end="")
print('{}'.format(secondWord))
```

A
A

AendA n A Ac Aw A Afb A A A n A Ac A A nA A A
A A n A AA fb A AendA n Afb A A A A A A A AfbA AA
A n A Ac wA A AA A AEV3APython AA
A

A A A AWA A A A A A Afb A An A y A A A A
A A A nAfb A AA A c A A A A AV A Ac A A
AfileA n A A Ac A
A
A

```
#!/usr/bin/env python3

from sys import stderr

# printing to the VSCode console
print('EV3 Python', file=stderr)

print('{} '.format(firstWord), end="", file=stderr)
print('{}'.format(secondWord), file=stderr)
```

A
A

A A AA wA A A A A AV A A A AAA A A AA
Aw AC AA A A Afb A n A AG Ay n AA n A A A
A A A A A A Afb AonForSeconds()Afb A A A Afb A A
Afb An AA An A Ay A A A A A A A A A A
A Aw A
A
A

Leave Logging in your code forever

A

A A Afb AA n yA n A A A A c A A A A wAA A
c n AA A w An A w A A A A A A A A A An wA A A
A y A AA A A A Aw c A A An A Afb A fb Aw Afb
A A A A A A fb A c n A A
A

A Ac A n A A AA c Afb A A A A A A A A Ac AA A A n A
A A A Afbdebug A A ATrue A AFalse A A c A A c A A A
A A A n A
A
A

```
def doSomething(debug):
```

```python
    if debug:
        print('doing something.', file = stderr)

def main():

    debug = True

    if debug:
        print('Start.', file = stderr)

    if debug:
        print('Finished.', file = stderr)

main()
```

A

A

A A A A A A A n A A A A n A A A A

n A fA fb n A A n A A A A c c A A A A c A A A w A A A A

A A A A fA A w A A n w A A n n A A A

A

A

Binary Numbers

A

C fb A A A A r A A A A A A n c A

A

A A A A A A A n A A A A A fb A A A w A A fb A A A

A n A A A A A n c A A w A A w A fb n A A A A A A A A

A n A A A A A A A A A G A y n A A n c A A A A A A A

A

Thousands	Hundreds	Tens	Units
1	2	3	4

A

A

A A fA A n c A A n A A A A r w A A A A **plus** A A

A **plus** A A A **plus** A fb A A A A A n A A A A A C A A A

n A A A A A n c A A A A A A n A c A A A n A A A A

A

A A fA w A A n A n c A A c A A A n A A A A A

A fb A r A A A A A fA A n A A A n w A A A A

A

10 to power of 3 = 1,000	10 to power of 2 = 100	10 to power of 1 = 10	10 to power of 0 = 1
1	2	3	4

A
A
A n c A A A r A A A y A A A y A A A y A A
A
n A A A w A b A A A f a A C A A n A A A A n A
A A A c A n A A A A A A A A n A A A w A A w A f a
A A A A n A A A A A f a A n A A A f b A
A
A n A n c A A A A A A A A c A n A A A c A A A A A A
A A A A A n A A A w A A A f b A A n A A A A w A A A
n A r w A A A w A A A n A w A A
A
A

2 to power of 3 = 8	2 to power of 2 = 4	2 to power of 1 = 2	2 to power of 0 = 1
1	1	0	1

A
A
A
A A A A A A n A A A n A r w A f a A n A A A A A A A A A
plus A A **plus** A A r A A
A
A

Using Binary numbers as Flags

A
A A A A f b A A c A n c A c A f y A A A A A c A f b A c A A A
A A A n A A A n c A A A c A n c A A A A A A c A
A
n A A A n A A A A
A
A

```
#!/usr/bin/env python3

ham =    0b0001
cheese = 0b0010
tomato = 0b0100
bread =  0b1000
```

```
mySandwich = ham + cheese + bread
```

A
A
A A Ac A A A A wA A A A nA A A n Ar w Afb A
AAr w A A A n Ar AA Ac Ar A AA A A r A nc A
A A Ac AA Afb AA A A A A A n A A n AA
A
A Afb mySandwich A A fb A
A
A

```
mySandwhich =    0b0001  (ham)
               + 0b0010  (cheese)
               + 0b1000  (bread)
               = 0b1011
```

A
A
w A A A n A A AA A A w A A n A nc A AA A Ac A A A
Ac AA
A
A

A
A
A
fb A A A w A Av A 0b1011 A An Afbw A An A A A A n A A
n A A A A A Ac A A AA A A A n A A A n AG A A
An A A A A AA A A n A An A A A A A Afb A A
A A n A n A
A
n n AA A A A An A n A A Afb A A A A
A
A

```
#!/usr/bin/env python3

ham = 0b0001
cheese = 0b0010
tomato = 0b0100
bread = 0b1000

mySandwich = ham + cheese + bread

print("Your sandwhich has ", end="")
```

```
if mySandwich & ham:
    print("ham ", end="")
if mySandwich & cheese:
    print("cheese ", end="")
if mySandwich & tomato:
    print("tomato ", end="")
if mySandwich & bread:
    print("bread ", end="")
```

```
Your sandwich has ham cheese bread
```

```
if mySandwich & ham:
    print("ham ", end="")
```

```
    0b1011  (mySandwich)
AND 0b0001  (ham)
  = 0b0001
```

A

```
     0b1011  (mySandwich)
  OR 0b0001  (ham)
   = 0b1011
```

A
A
A A A A A fA n c A A A Ac An A
A
A

Conditioning Debug Statements with Bit Masking

A
An A Ac A c A A c A A A A w A fA A Aw A A fA A
n A A A A A A Ac A n Ac A An A AC An A
A A A A A A
A
C A A A A fA A A Ac A A A A A A A A A A A A n A A
A A A A A y A A A Afb n A
A
A

```
DEBUG_NONE                                  = 0
DEBUG                                       = 0b00000001

DEBUG_THREAD_LIFECYCLE                      = 0b00000010


DEBUG_MOVEMENT_ROTATION_STARTING_POSITION   = 0b00000100
DEBUG_MOVEMENT_ROTATION_CURRENT_POSITION    = 0b00001000
DEBUG_MOVEMENT_ROTATION_FINAL_POSITION      = 0b00010000
```

A
A
A A A A Aw ADEBUG_NONEA ADEBUG A A A A A A A A A
A ffbAA ADEBUG_THREAD_LIFECYCLEAfb A Ac A A A A A fb A fA A A
fb n A A A n AA A n Afb Aw An Ac A A A A A A fA A
w A A A ffb A w A fA AAA
A
A A A A Ac AA A A A A A A A c Aw A fA A w Afb A A
A A A A A A fA A fbAn Ac fb A A An AA A n A
A A A A A A A A fA A fbAn A An A A A A n A A
fb A n c A fA A
A
A

```python
def driveForXRotations(debug, stop, rotations, speed):

    motorLeft = LargeMotor(constants.OUTPUT_LARGE_MOTOR_LEFT)
    tank_pair = MoveTank(constants.OUTPUT_LARGE_MOTOR_LEFT,
                         constants.OUTPUT_LARGE_MOTOR_RIGHT)

    rotationB = motorLeft.position

    tank_pair.on(left_speed=speed, right_speed=speed)
```

```
    while motorLeft.position < rotationB + (rotations * 360):

        if stop():
            break

    tank_pair.off()
```

A
AA
A
G A A A n A A A A fb A fa fb n A A A n AAA w A
A A A A AA A A A fa fb A A A A A A A c A fa A
c A A fb A A A A A fa fb A fa A A A n A A AA A
A A A A fb A n A n A A A A A A A
n AA w A A A A A if not stop():AA
A
A n A A A Ac A
A
A

```
def driveForXRotations(debug, stop, rotations, speed):

    if debug & DEBUG and debug & DEBUG_THREAD_LIFECYCLE:
        print("Start driveForXRotations({}, {}), thread {}.".format(rotations, speed,
              threading.current_thread().ident), file=stderr)

    motorLeft = LargeMotor(constants.OUTPUT_LARGE_MOTOR_LEFT)
    tank_pair = MoveTank(constants.OUTPUT_LARGE_MOTOR_LEFT,
                         constants.OUTPUT_LARGE_MOTOR_RIGHT)

    rotationB = motorLeft.position
    tank_pair.on(left_speed=speed, right_speed=speed)

    while motorLeft.position < rotationB + (rotations * 360):

        if stop():
            if debug & DEBUG and debug & DEBUG_THREAD_LIFECYCLE:
                print("Kill driveForXRotations({}, {}), thread {}.".format(rotations,
                      speed, threading.current_thread().ident), file=stderr)
            break

    tank_pair.off()

    if not stop():
        if debug & DEBUG and debug & DEBUG_THREAD_LIFECYCLE:
            print("End driveForXRotations({}, {}), thread {}.".format(rotations, speed,
                  threading.current_thread().ident), file=stderr)
```

A

AA
A c A A A A A A A A A A A fb A A A A A A A A A
fb A A
A
A

```
debugFlags = DEBUG | DEBUG_THREAD_LIFECYCLE
driveForXRotations(debugFlags, stop, 2, 50)
```

A
A
A fb A A A Ac fb A Aprint()A n AA Afb A Adebug&ADEBUGA
A A Afb A A A c An A A AA A A Adebug &
DEBUG_THREAD_LIFECYCLEAA A A Afb A Afb A Ac A AA
A
y A A A A A A Afb A AA A A n c A AA r A fb A
A A A n Ac A A n AA A A A An A A Ac A
fb A A A Afb A
A
A

```
Start driveForRotations(2, 50), thread 01080.
End driveForRotations(2, 50), thread 01080
```

A
A
A A A A A A A A A A A A Ac AA A A A
A An Afb A A A A A A
A
A

```
def driveForXRotations(debug, stop, rotations, speed):

    if debug & constants.DEBUG and debug & constants.DEBUG_THREAD_LIFECYCLE:
        print("Start driveForXRotations({}, {}), thread {}".format(rotations, speed,
              threading.current_thread().ident), file=stderr)

    motorLeft = LargeMotor(constants.OUTPUT_LARGE_MOTOR_LEFT)
    tank_pair = MoveTank(constants.OUTPUT_LARGE_MOTOR_LEFT,
                         constants.OUTPUT_LARGE_MOTOR_RIGHT)

    rotationB = motorLeft.position

    if debug & DEBUG and debug & DEBUG_MOVEMENT_ROTATION_STARTING_POSITION_POSITION:
        print("> Starting position {}". format(rotationB), file = stderr)

    tank_pair.on(left_speed=speed, right_speed=speed)

    while motorLeft.position < rotationB + (rotations * 360):

        if debug & DEBUG and debug & DEBUG_MOVEMENT_ROTATION_CURRENT_POSITION:
```

```
            print("> Current position {}". format(motorLeft.position), file = stderr)

        if stop():
            if debug & constants.DEBUG and debug & DEBUG_MOVEMENT_ROTATION_FINAL_POSITION:
                print("> Final position {}". format(rotationB), file = stderr)

            if debug & constants.DEBUG and debug & constants.DEBUG_THREAD_LIFECYCLE:
                print("Kill driveForXRotations({}, {}), thread {}.".format(rotations,
                      speed, threading.current_thread().ident), file=stderr)
            break

    tank_pair.off()

    if debug & constants.DEBUG and debug & DEBUG_MOVEMENT_ROTATION_FINAL_POSITION:
        print("> Final position {}". format(motorLeft.position), file = stderr)

    if not stop():

        if debug & constants.DEBUG and debug & constants.DEBUG_THREAD_LIFECYCLE:
            print("End driveForXRotations({}, {}), thread {}.".format(rotations, speed,
                  threading.current_thread().ident), file=stderr)
```

A
AA
A A A A A A fA An A Ac A w Ac A A A A DEBUG A
DEBUG_MOVEMENT_ROTATION_STARTING_POSITIONA A
DEBUG_MOVEMENT_ROTATION_FINAL_POSITIONA n A A Ac A
A
A

```
debugFlags = DEBUG | DEBUG_MOVEMENT_ROTATION_STARTING_POSITION |
                     DEBUG_MOVEMENT_ROTATION_FINAL_POSITION
driveForXRotations(debugFlags, stop, 2, 50)
```

A
A
AA
A A A A A A A A DEBUG_MOVEMENT_ROTATION_CURRENT_POSITIONA
fb AA A A A A A A A A A A A w Ac A AA A A
r A A
A
A

```
debugFlags = DEBUG | DEBUG_MOVEMENT_ROTATION_STARTING_POSITION |
                     DEBUG_MOVEMENT_ROTATION_CURRENT_POSITION |
                     DEBUG_MOVEMENT_ROTATION_FINAL_POSITION
driveForXRotations(debugFlags, stop, 2, 50)
```

A

A

C A A A A A A wA AflAfb A A Ac n AAc A AA A A A A A A A Ac A c AA AflA A A A A An A A A A A A wA An Afb A c A A An A

A

A AA AflA A A A A A A A Ac AA Afb A A A AA Afb A ADEBUG_MOVEMENT_ROTATION_ALL A An A A w A Afb A AA A A A A A A A ADEBUG_MOVEMENT_ROTATION_XXXAfb Ac An A fb A A Afb A

A

flA Afb A An A A A Aw A A A AAfb A ACC A AAfb A A C W A A A AflA Ac A A AA Ac wA A A wA Aw Afb A A

A

A

```
DEBUG_NONE                                     = 0
DEBUG                                          = 0b00000001

DEBUG_THREAD_LIFECYCLE                         = 0b00000010


DEBUG_MOVEMENT_ROTATION_STARTING_POSITION      = 0b00000100
DEBUG_MOVEMENT_ROTATION_CURRENT_POSITION       = 0b00001000
DEBUG_MOVEMENT_ROTATION_FINAL_POSITION         = 0b00010000

DEBUG_MOVEMENT_ROTATION_ALL                    = DEBUG_MOVEMENT_ROTATION_STARTING_POSITION |
                                                 DEBUG_MOVEMENT_ROTATION_CURRENT_POSITION |
                                                 DEBUG_MOVEMENT_ROTATION_FINAL_POSITION


DEBUG_EVERYTHING                               = 0b11111111
```

A

A

A A AA

A

A A nA A A nc AflAfb A A A n A A A c A fb A Afb A Ac A nc A A c c A A A A A A AAA A A A A flfb Ac A A AflA A A AA

A

A A w A Aw A Ac A fb A A A A A A Ac AA Aw AA A A A A A AflAA wA nc A A A A A A A A A A An A A A AA r A

A

A

```
DEBUG_NONE                                     = 0


DEBUG                                          = 2 ** 0
DEBUG_THREAD_LIFECYCLE                         = 2 ** 1


DEBUG_MOVEMENT_ROTATION_STARTING_POSITION      = 2 ** 2
DEBUG_MOVEMENT_ROTATION_CURRENT_POSITION       = 2 ** 3
DEBUG_MOVEMENT_ROTATION_FINAL_POSITION         = 2 ** 4
```

A