3. Loops and if-Statements

Introduction:

for-loops, while-loops, and if-statements allow greater flexibility in writing programs. This lecture covers using these in Micropython. A quick summary of loops is as follows.

```
# for-loops
for i in range(1,6):
    d1 = i
for j in [1,2,3,5,7,9]:
    d2 = j

# while-loopps
t = 0
while(t < 10):
    t += 0.01

while(1):
    print('infinte loop')

#if statements
if(a > b):
    print('a > b')

if(a != b):
    print('a is not equal to b')
else:
    print('a is equal to b')

if(a > b):
    print('a is greater than b')
elif(a == b):
    print('a equals b')
else:
    print('a is less than b')
```

For-Loops

for-loops behave about the same as they behave in Matlab:

- A variable is required for the loop (in the example below)
- The variable increments as you go through the loop
- The looping stops once you reach the end

There are some differences, however

- A colon is required - this marks the beginning of the for-loop
- Subsequent lines of code *must* be indented. This indentation marks the lines of code within the loop
- To signify the end of the loop, go back to the original indentation.
- (note: 4 spaces are standard indentation in Micropython - although the compiled accepts anything)

The indentation is important

- Indentation signifies where the loop starts and ends
- Inconsistent indentation will cause an execution error: the compiler doesn't know why the indentation was changed and doesn't know what to do.

For example, the following are two separate for-loops:

- The first for-loop finishes then the second one begins:

```
for i in range(1,6):
    d1 = i
for j in range(1,4):
    d2 = j
```

In contrast, this would be an example of nested loops:

```
for i in range(1,6):
    d1 = i
    for j in range(1,4):
        d2 = j
        roll = d1 + d2
```

In Micropython, you must have a statement within the loop. If you don't want to do anything, use the *pass* command: it behaves like a *nop* statement. As an example, one way to kill time is to count just for the sake of counting:

```
for i in range(0,100):
    for j in range(0,100):
        pass
```

Here, the *pass* command is executed 100x100 times (nested loops)

Another difference with MicroPython and Matlab is the *range()* statement

- The range starts at 0 (same as Matlab), but
- Runs while i < 5 (vs. <= 5 for Matlab)


```
for i in range(0,5):
    x = i*i
    print(i, 'squared = 'x)
```

Thonny Shell (Micropython)

```
>>>
0 squared =  0
1 squared =  1
2 squared =  4
3 squared =  9
4 squared =  16
```

If you want to use Matlab syntax where you stop at 5 (rather than 4.999), change this to 5.01

```
for i in range(0,5.01):
    x = i*i
    print(i, 'squared = 'x)
```

Thonny Shell (Micropython)

```
>>>
0 squared =  0
1 squared =  1
2 squared =  4
3 squared =  9
4 squared =  16
5 squared =  25
```

If you add a 3rd term in the *range()* statement, the 3rd term is the step size. For example, make the step size equal to two:

```
for i in range(0,10.1,2):
    x = i*i
    print(i, 'squared = 'x)
```

Thonny Shell (Micropython)

```
>>>
0 squared =  0
2 squared =  4
4 squared =  16
6 squared =  36
8 squared =  64
10 squared =  100
```

You can also step through an array. For example, to find the squares of prime numbers:

```
prime = [1,2,3,5,7,9]
for i in prime:
    x = i*i
    print(i, 'squared = 'x)
```

Thonny Shell (Micropython)

```
>>>
0 squared =  0
2 squared =  4
4 squared =  16
6 squared =  36
8 squared =  64
10 squared =  100
```

For-Loop Example: Timer2 Interrupts

Back in ECE 376, we talked about Timer2 interrupts. To generate a given frequency using Timer2 interrupts, you want to interrupt every N seconds with

$$N = \left(\frac{10{,}000{,}000}{2 \cdot Hz} \right) \text{ clocks}$$

N is in turn set with three constants:

N = A*B*C

where

- A = 1..16
- B = 1..256
- C = 1, 4, or 16

The challenge is finding {A, B, C} so that N is close to its desired value.

With for-loops, you can go through every combination of {A, B, C} and pick the one with the smallest error. For example, to set the freuquencey to 327.63Hz:


```
Hz = 327.63
N0 = 10_000_000 / (2*Hz)
print('Target N = ',N0)
A, B, C = 0, 0, 0
MinError = 9999
for a in range(1,17):
    for b in range(1,257):
        for c in [1, 4, 16]:
            N = a*b*c
            Error = abs(N - N0)
            if(Error < MinError):
                A = a
                B = b
                C = c
                MinError = Error
print('A = ',A)
print('B = ',B)
print('C = ',C
print('N = ',A*B*C)
```

Thonny Shell (Micropython)

```
>>>
N = 15261.12
A = 6
B = 159
C = 16
N = 15264
```

The closest you can come is A*B*C = 15,264 (off by 3.12 clocks)

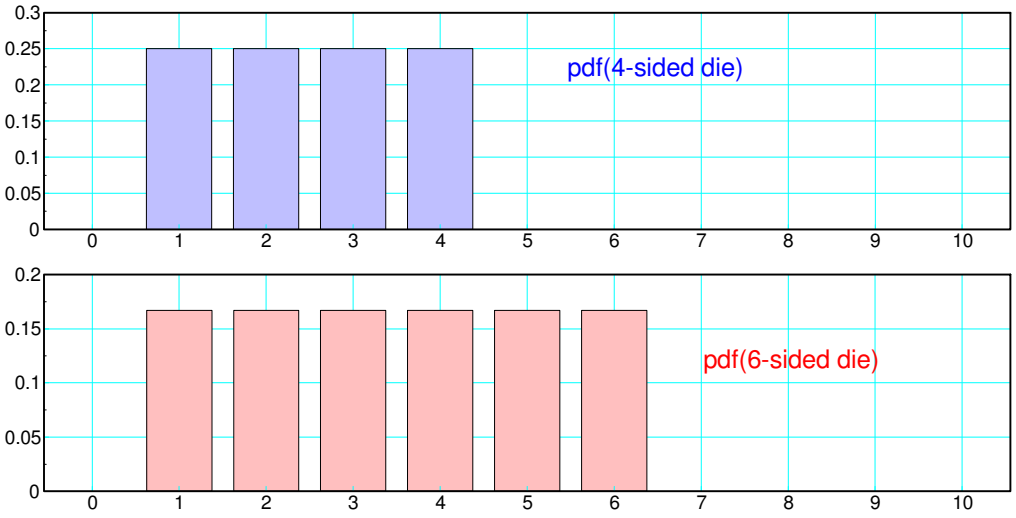
For-Loop Example: Creating Arrays

As an example of using for-loops, create an array which indicated the probability of getting the numbers 0..10 when rolling

- A 4-sided die, and
- A 6-sided die

The array should like the following:

k (die roll)	0	1	2	3	4	5	6	7	8	9	10
d4	0	1/4	1/4	1/4	1/4	0	0	0	0	0	0
d6	0	1/6	1/6	1/6	1/6	1/6	1/6	0	0	0	0



pdf for a 4-sided and 6-sided die

In Micropython, there are a couple of ways of doing this (unformatted output):

Option #: No Finesse

```
d4 = [0,1/4,1/4,1/4,1/4,0,0,0,0,0,0]
d6 = [0,1/6,1/6,1/6/1/6,1/6,1/6,0,0,0,0]
```

Option 2: Use a for-loop

```
d4 = [0]*10
for ki in range(1,4.1):
   d4[k] = 1/4
d6 = [0]*10
for k in range(1,6.1):
   d6[k] = 1/6
```

Option #3: Use a subroutine

something we'll cover shortly

You can also format the output:


```
d4 = [0]*11
for i in range(1,4.01):
   d4[i] = 1/4
d6 = [0]*10
for k in range(1,6.01):
   d6[k] = 1/6

print('  k    d4    d6')
for k in range(0,11):
    print('{: 3.0f}'.format(k), '{: 3.0f}'.format(d4[k]), '{:
3.0f}'.format(d6[k]),
```

Thonny Shell (Micropython)

```
>>>
 k       d4      d6
 0      0.000   0.000
 1      0.250   0.167
 2      0.250   0.167
 3      0.250   0.167
 4      0.250   0.167
 5      0.000   0.167
 6      0.000   0.167
 7      0.000   0.000
 8      0.000   0.000
 9      0.000   0.000
10      0.000   0.000
```

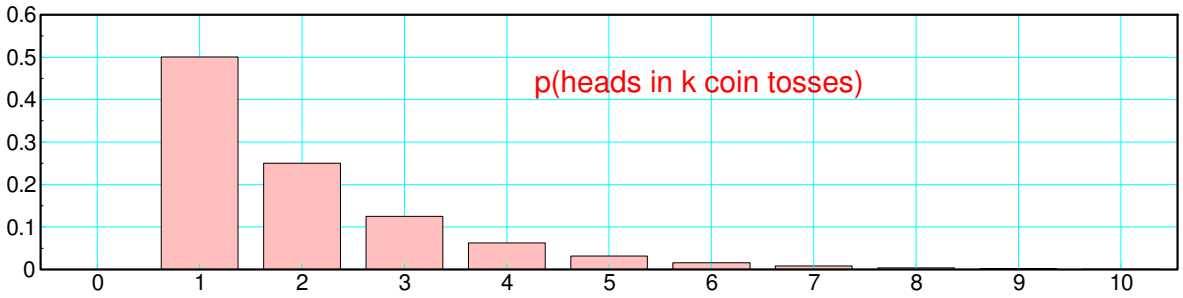
While-Loops

A while loop keeps going

- As long as a condition holds, or
- Until you encounter a *break* statement

For example, the probability of flipping a coin k times before you get a heads (exponential distribution) is:

$$p(k) = \left(\frac{1}{2}\right)\left(\frac{1}{2}\right)^{k-1} u(k-1)$$



[This series goes out to infinity - which is a problem. If you truncate the series using a for-loop, you get a fixed number of terms - such as ten terms (below)

```
k = [0]
p = [0]
for i in range(1,11):
   k.append(i)
   p.append(0.5 * ( 0.5 ** (i-1) )
print('  k     p(k)')
for i in range(0,11):
    print('{: 3.0f}'.format(k[i]), '{: 3.0f}'.format(p[i]))
```

Thonny Shell (Micropython)

```
>>>
 k       p(k)
 0      0.000
 1      0.500
 2      0.250
 3      0.125
 4      0.063
 5      0.031
 6      0.016
 7      0.008
 8      0.004
 9      0.002
10      0.001
```

If you use a while loop, you can stop as soon as p(k) < 0.01

```
p = [0]
x = 0.5
k = 0
while (x > 0.01):
   k += 1
   x = 0.5 * (0.5 ** (k-1))
   p.append(x)
for k in range(0,len(p)):
    print('{: 3.0f}'.format(k), '{: 3.0f}'.format(p[k]))
```

Thonny Shell (Micropython)

```
>>>
 k       p(k)
 0      0.000
 1      0.500
 2      0.250
 3      0.125
 4      0.063
 5      0.031
 6      0.016
 7      0.008
```

Another common use of while statements is to set up an infinite loop

```
while(1):
    X = float(input('X = '))
    Y = X*X
    print('The square of ',X,'is ',Y)
```

Thonny Shell (Micropython)

```
X = 3
The square of 3 is 9
X = 4.2
The square of 4.2 is 17.64
```

Press the Stop symbol to break out of an infinite loop

If Statements

If statements precede a set of commands that are executed one time if a condition is true. Conditional statements are:

```
X > Y     X is greater than Y
X < Y     X is less than Y
X >= Y    X is greater than or equal to Y
X == Y    X is equal to Y
X != Y    X is not equal to Y
&         logical and
|         logical or
^         logical xor
```

Indentation indicates the statements that are within the for loop.

```
if(x>y):
    print('x is greater than y')
if(x<y):
    print('x is less than y')
if(x==y):
    print('x is equal to y')
```

else indicates instructions to execute if the if-statement is false

```
if(x>y):
    print('x is greater than y')
else:
    print('x is less than or equal to y')
```

elif is an else-if statement

```
if(x>y):
    print('x is greater than y')
elif(x<y):
    print('x is less than y')
else(x==y):
    print('x is equal to y')
```

One place where else-if is useful is when you have different bands. For example, the following code is equivalent:

```
# Option 1
if(T>40):
    print('Really hot: T > 40')
if( (T>30)&(T<=40)):
    print('Hot: 30<T<40)')
if( (T>20)&(T<=30)):
    print('Comfortable: 20<T<30')
if( (T>10)&(T<=20)):
    print('Cool: 10<T<20')
```

or using else-statements

```
# Option 2
if(T>40):
    print('Really hot: T > 40')
elif( T>30):
    print('Hot: 30<T<40)')
elif( T>20):
    print('Comfortable: 20<T<30')
elif( T>10):
    print('Cool: 10<T<20')
else:
    print('Chilly: T < 10')
```

With if-statements, you can create probability density functions more efficiently. For example, another way to create the pdf for rolling a 4-sided and 6-sided die is:


```
d4 = []
d6 = []
for k in range(0,11):
    if( (k>=1) & (k<=4) ):
        d4.append(1/4)
    else:
        d4.append(0)
    if( (k>=1) & (k<=6)):
        d6.append(1/6)
    else:
        d6.append(0)
for k in range(0,11):
    print('   k    d4     d6')
for k in range(0,11):
    print('{: 3.0f}'.format(k), '{:
3.0f}'.format(d4[k]), '{: 3.0f}'.format(d6[k]),
```

Thonny Shell (Micropython)

```
>>>
 k       d4      d6
 0     0.000   0.000
 1     0.250   0.167
 2     0.250   0.167
 3     0.250   0.167
 4     0.250   0.167
 5     0.000   0.167
 6     0.000   0.167
 7     0.000   0.000
 8     0.000   0.000
 9     0.000   0.000
10     0.000   0.000
```

With if-statements, you can also do convolution. The probability of the sum of a 4-sided and 6-sided die are:

$$y = d4 + d6$$

$$p(y = k) = \sum_{n=0}^{\infty} d4(n) \cdot d6(k - n)$$

In Micropython, clipping the summation where the arrays are out of bounds

- (k-n) < 0 and (k-n) >= 12)


```
d4 = [0]
d6 = [0]
y = [0]
for k in range(0,12):
    y.append(0)
    if(k<=4):
        d4.append(1/4)
    else:
        d4.append(0)
    if(k<=6):
        d6.append(1/6)
    else:
        d6.append(0)
for k in range(0,12):
    y[k] = 0
    for n in range(0,12):
        if( (k-n>0) & (k-n)<12) ):
            y[k] += d4[n]*d6[k-n]
print('  k  d4+d6'
for k in range(0,12):
    print('{: 3.0f}'.format(k), '{: 3.0f}'.format(y[k]) )
```

Thonny Shell (Micropython)

```
>>>
 k      d4+d6
 0      0.000
 1      0.000
 2      0.042
 3      0.083
 4      0.125
 5      0.167
 6      0.167
 7      0.167
 8      0.125
 9      0.083
10      0.042
11      0.000
```

The probability of the sum of a d4 and d6 is 3 is 0.083