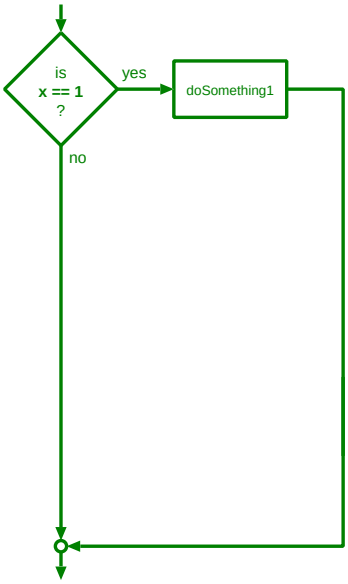
Information Processing 1 — Week 6 exercises

Complete each exercise on your own laptop before the end of the exercise class. When you finish an exercise, **immediately** ask an instructor to check your answer. Answers checked after the end of the class will have a 50% penalty.

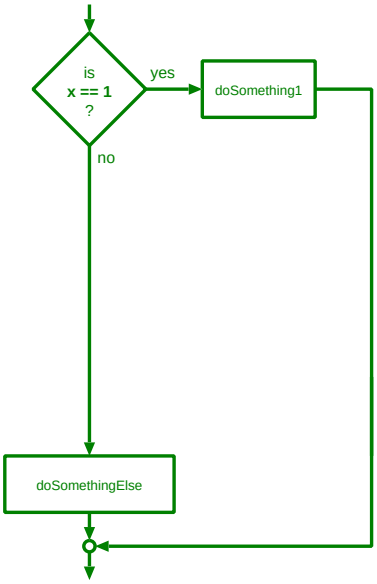
Do not struggle for more than a few minutes with anything you cannot solve. If you are stuck or have any questions then **raise your hand** and an instructor will help you.

1 Conditional statements [4 points]

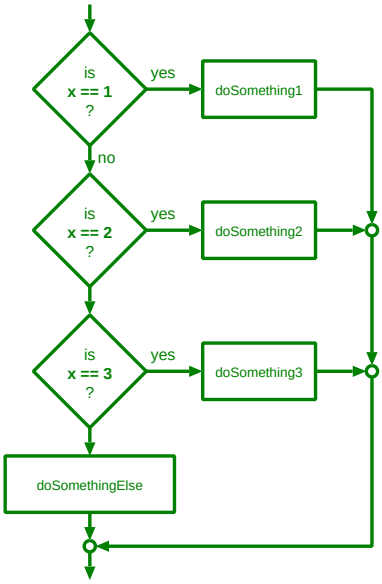
A conditional statement has a mandatory '`if`' clause with a condition and some statements, followed by zero or more optional '`elif`' clauses each with their own condition and statements, and an optional final '`else`' clause with some statements (but no attached condition).



`if` *condition1*:
    statements to execute if condition1 is true



`if` *condition1*:
    statements to execute if condition1 is true
`else`:
    statements to execute if condition1 was not true



`if` *condition1*:
    statements to execute if condition1 is true
`elif` *condition2*:
    statements to execute if condition2 is true
`elif ...`
`elif ...`
`else`:
    statements to execute if none of the above conditions are true

The first `condition1` is evaluated and, if true, its statements (only) are executed and the entire 'if-elif(s)-else' conditional statement is complete (control continues with the rest of the program). Otherwise the next `condition2` (if present) is evaluated and, if true, its statements are executed and the entire conditional statement is compete. If none of the conditions are true, the '`else`' statements (if present) are executed.

1.1 Even, odd, divisible by n

The following function prints whether `n` is even or odd:

```
def evenOdd(n):
    if n % 2:
        print("odd")
    else:
        print("even")
```

Part 1. Modify the function to return one of the strings `"even"` or `"odd"`, instead of printing it. Test your function by printing the results for the first 10 non-negative integers:

```
for i in range(10):
    print(i, evenOdd(i))
```

Part 2. Write another function `isDivisibleBy(n, m)` that returns `True` if `n` is *exactly* divisible by `m`, `False` if it is not. Rewrite `evenOdd(n)` to use `isDivisibleBy(n, m)`.

Part 3. Use `isDivisibleBy(n, m)` to write a loop that prints *only* the numbers between 0 and 100 that are divisible by 7: 0, 7, 14, 21, etc. (Numbers not divisible by 7 should not be printed at all.)

1.2 Age groups

Let's use a person's age to place them into a group as follows:

age (years)	*age group*
0 – 5	baby
6 – 17	school
18 – 21	student
22 – 64	employed
65 –	retired

Write a function `ageGroup(age)` that returns a string describing the group corresponding to an `age` in years. (Use the same group names as shown in the table above.)

Test your function using a loop that calls it once for each `age` between 0 and 69.

Hint: If you do not know immediately how to write the function then you can develop it incrementally, as follows.

Start with this simple function that uses an '`if`' statement to return `"baby"` if `age` is less than 6.

```
def ageGroup(age):
    if age < 6:
        return "baby"
```

The next increment is to add an `if` or `elif` to the function so that it returns `"school"` if `age` is below 18. Test your function again, with various ages. Continue by adding more code that returns `"student"`

and `"employed"` according to `age`. Each time you add new code, test your function again. Finally add an `else`: (or just a final '`return`') that gives the result `"retired"` for any `age` over 64.

▷▷ Ask an instructor to check and record your work **now** (before you run out of time).

2 Recursive functions [3 points]

Let's implement some well-known recursive mathematical functions. If you already know how to do this then you can skip directly to Section 2.1. If you are not sure how to do this, let's practice by incrementally developing a recursive function that calculates the factorial of a number.

$$\text{factorial}(n) = \begin{cases} n \times \text{factorial}(n-1) & \text{if } n > 0 \\ 1 & \text{otherwise} \end{cases}$$

Using incremental development we can start by considering only the trivial 'otherwise' case. Write a function `factorial(n)` that returns `1` if `n` is less than `1`, otherwise it returns `n`. Test your function.

```
factorial(0)    # 1
factorial(1)    # 1
factorial(2)    # 2
factorial(3)    # 3
```

Modify your function so that instead of returning `n` it returns `n` multiplied by the factorial of `n-1`. Test your function.

```
factorial(0)    # 1
factorial(3)    # 6
factorial(6)    # 720
```

2.1 Sum of counting numbers

Write a function `sumOfCount(n)` that returns the sum of the counting numbers from `1` to `n`.

$$\text{sumOfCount}(n) = \begin{cases} n + \text{sumOfCount}(n-1) & \text{if } n > 0 \\ 0 & \text{otherwise} \end{cases}$$

Test your code by printing the results for 0 to 9: 0 1 3 6 10 15 21 28 36 45

2.2 Fibonacci numbers

The Fibonacci numbers belong to the sequence in which each number is the sum of the previous two numbers, starting with 0 and 1: 0 1 1 2 3 5 8 13 21 34 55 89 ...

Write a function `fibonacci(n)` that returns the n^{th} Fibonacci number. (Consider the 1st in the sequence to be the first of the two '1's.)

$$\text{fibonacci}(n) = \begin{cases} \text{fibonacci}(n-1) + \text{fibonacci}(n-2) & \text{if } n > 2 \\ 1 & \text{otherwise} \end{cases}$$

(Note that this is an example of a 'doubly recursive' function.)

2.3 Greatest common divisor

The greatest common divisor (GCD) of two numbers is the largest integer that exactly divides both of them. For example:

24 can be divided by:	2	3	4	6		8	12	
42 can be divided by:	2	3		6	7		14	21

The common divisors of 24 and 42 are 2, 3, and 6. The GCD of 24 and 42 is therefore 6.

Write a function `gcd(a, b)` using the Euclidian method, as follows:

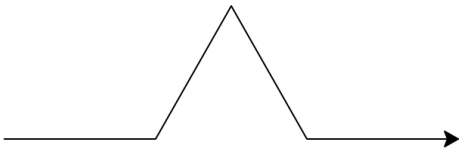
$$\gcd(a,b) = \begin{cases} \gcd(b, a \bmod b) & \text{if } b \neq 0 \\ a & \text{if } b = 0 \end{cases}$$

Note that in Python the modulus operation '$a \bmod b$' is written using the percent symbol as '`a % b`'.

▷▷ Ask an instructor to check and record your work **now** (before you run out of time).

3 Recursive geometry [2 points]

To draw a Koch curve whose length is x you should:



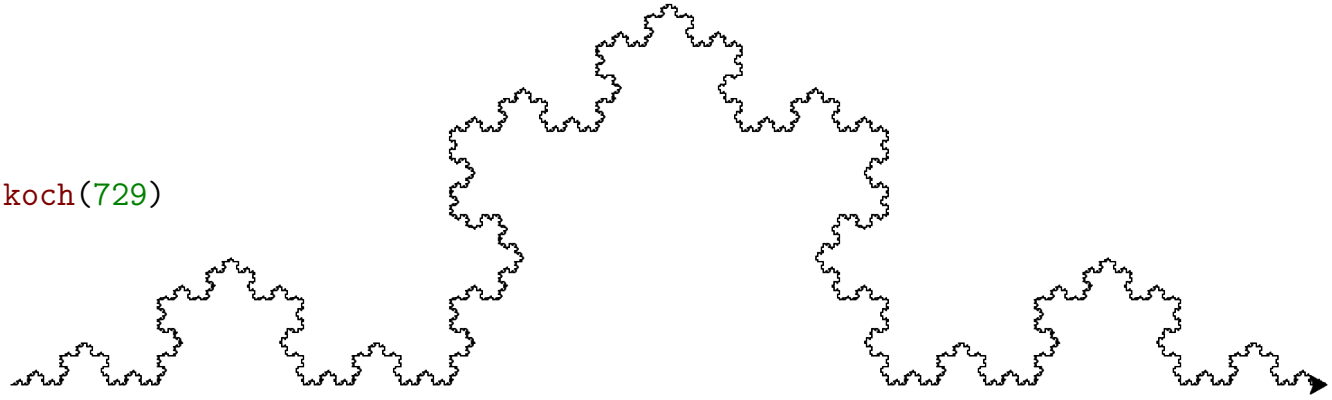
- draw a Koch curve with length $x \div 3$,
- turn left 60 degrees,
- draw a Koch curve with length $x \div 3$,
- turn right 120 degrees,
- draw a Koch curve with length $x \div 3$,
- turn left 60 degrees
- draw a Koch curve with length $x \div 3$,

Note that this definition is recursive. To stop the recursion continuing indefinitely, the exception to the above rule is when $x < 3$. In that case, just draw a straight line of length x.

Write a function called `koch(x)` that draws a Koch curve using the `turtle` module.

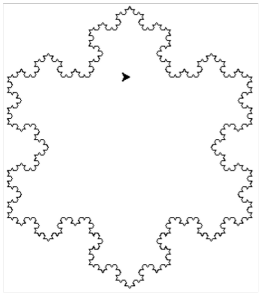
Hint: to exit immediately from the function at any point within it, use a `return` statement.

`koch(729)`



▷▷ Ask an instructor to check and record your work **now** (before you run out of time).

4 Snowflakes [1 point]



Write a function called `snowflake(length)` that uses your `koch()` function to draw a snowflake.

Hint: The top part of the snowflake shown on the right, spanning the width of the snowflake, is the same single Koch curve that was illustrated in the previous question.

▷▷ Ask an instructor to check and record your work **now** (before you run out of time).

Challenges

Complete all the challenges to earn one bonus point.

5 Fundamental algorithms: printing integers

When you use '`print(n)`' to print a number you do not see any of the complexity underlying the conversion of a numerical value `n` into the sequence of individual digits that represents it in written form. Let's fix that by exploring how integers are actually printed.

5.1 Separating the units column from the rest of the digits

Write a function called `printNum(n)` that prints the units (rightmost) digit in the decimal representation of the number `n`, followed by the remaining digits (all *but* the units column). For example, `printNum(3210)` should print:

```
0
321
```

Note that if you were to call `printNum(321)` next, it would print `1` followed by `32`.

5.2 Printing all the digits of an integer, backwards

Modify `printNum(n)` so that it prints the units digit of `n` and then, if there are any remaining digits to to the left of the units, it calls itself recursively to print them. For example, `printNum(3210)` should print:

```
0
1
2
3
```

Note that the only reason the digits come out backwards is because your function chooses to print the units digit *before* dealing with the remaining digits (if any).

5.3 Printing all the digits of an integer, forwards

Reorder the statements in `printNum(n)` so that the remaining digits (if any) are dealt with first and printing the units digit happens last. For the input `3210` your function should now be printing this:

```
3
2
1
0
```

5.4 Eliminating the newline characters between digits

You can suppress the newline when printing a value `x` like this:

```
print(x, end="")
```

Use this technique in `printNum(n)` so that the digits are all printed on one line.

5.5 Printing in bases other than 10

Our `printNum` function prints its parameter in decimal (base 10). Add a second parameter `b` to `printNum(n, b)` that specifies the number base to use when printing `n`. Test your function with:

```
printNum(3210, 10)   # 3210
printNum(255, 8)     # 377
printNum(42, 2)      # 101010
```

5.6 Specifying the minimum number of digits to print

Add a third parameter `w` to `printNum(n, b, w)` that specifies the minimum number of digits to print (i.e., the *width* of the output).

Hint: If there are no more 'remaining' digits, but `w` digits have not yet been printed, continue printing recursively until `w` digits have been printed.

Test your function with:

```
printNum(3210, 10, 0)   # 3210
printNum(255, 8, 3)     # 377
printNum(255, 8, 4)     # 0377
printNum(42, 2, 8)      # 00101010
```

5.7 Specifying the character to use when padding

The leading '0's that you are printing are called 'padding'. There is nothing special about '0' and other characters could be used to provide the padding.

Add another parameter `p` to `printNum(n, b, w, p)` that specifies the padding character to use when there are no more remaining digits but `w` characters have not been printed.

Test your function with:

```
printNum(3210, 10, 8, ".")  # ....3210
printNum(255, 8, 8, " ")    #      377
printNum(42, 2, 8, "0")     # 00101010
```

5.8 Sensible defaults for base, width, and padding

Python lets you specify default values for parameters when there is no corresponding argument value given during the function call. The syntax is to 'assign' the desired default value to the parameter inside the parameter list. For example:

```
def add(a=3, b=4):
    return a + b

print(add(43, 21))  # 64
print(add(38))      # 42
print(add())        # 7
```

Modify `printNum` so that the default base is 10, default width is 1, and default padding character is a space.

▷▷ Ask an instructor to check and record your work **now** (before you run out of time).