RECURSION is

(1) a powerful approach to problem solving, where the problem is solved by reducing it to a simpler version of the original problem (and combining those solutions)

(2) a function that calls itself (to solve the simpler version of the original problem)

EXERCISE 1/DEMO: Develop a (RECURSIVE) function that has one parameter, n, a non-negative integer, and returns a string which is the digits of n in reverse order (i.e., starting with the low-order digit). For example

```python
>>> reverse_int(123)
'321'
```

DEVELOP THE (RECURSIVE) ALGORITHM:

(a) What is the simplest example? What is a solution?

(b) Next – try the next simplest example, e.g., 2-digit number, 12. What is a solution?

(c) Next – look at a slightly more complex example, e.g., a 3-digit number, 123. How can we solve this? (HINT: Think about solving it in terms of the simpler examples.)

(d) How about 1234? (HINT: Think about solving it in terms of the simpler examples.)
(e) Let’s start turning this into Python code:

(0) Write the function header, docstring (type contract, brief description, simple examples, and return statement:

(1) What is the simplest example? \[n \leq 9, \text{i.e., single digit}\] [just return n as a string]

(2) Next – 2-digit number, 34: [get lowest order digit, 4, and put it at the beginning of the string, then concatenate remaining digit: ‘4’ + ‘3’]

(3) Next – 3-digit number, 345: [get lowest order digit, 5, and put it at beginning of string. Then ... it’s the 2-digit case! ‘5’ + (‘4’ + ‘3’)]

(4) Next – 4 digit number:
Putting all of the Python code together:

HAND TRACE TO CHECK CODE. DRAW THE ACTIVATION RECORD FOR EACH FUNCTION CALL.
Lab Exercise 2

**Palindrome – recursive function:** Given a string, s, determine if it is a palindrome (reads the same backwards and forwards, e.g., 'kayak', 'racecar', 'eye'); return True if s is a palindrome and False otherwise. For example,

```python
>>> isPal('kayak')
True
>>> isPal('hello')
False
```

(a) Write out a simple example (or two or three) for palindrome:

(b) Consider the simplest case:

(c) Consider the next simplest case:

(d) Consider the next simplest case:

(e) Consider slightly longer strings – describe their solutions in terms of the simple cases
(f) Write the function header, docstring (type contract, brief description, examples from (a) - (d)), and return statement for function palindrome:
(g) Complete the code for the palindrome function (use only tools in current Python "toolkit"): HINT: Write the code starting with the code for the simplest example, then add the code for the next simplest example, and so on.

(h) Generate additional test cases for palindrome. Remember that CIS 210 test cases are always valid (per type contract and brief description) INPUT. Make sure test cases cover (1) normal examples, (2) edge/boundary examples, (3) different types of expected output, (4) different types of expected input. Are any changes to code needed?
More recursion practice:

(0) Write three functions that compute the sum of the numbers in a given integer (i.e., do not use strings) using a for-loop, a while-loop, and recursion. The functions should return the sum of the numbers.

(1) Rewrite isPal to implement an iterative algorithm. NOTE: the docstring and return statement of isPal should not change. Also, consider glass box testing: are additional tests needed to adequately exercise all of the code?

(2) Develop and implement a recursive algorithm for determining the value of integer j raised to the power of integer k. For example, powr(4, 3) → 64

(3) Develop and implement a recursive algorithm for determining the greatest common divisor (GCD) of two integers, a and b. For example, gcd(21, 15) → 3

Hint: One way to find the GCD of two numbers is Euclid’s algorithm, which is based on the observation that if r is the remainder when a is divided by b, then gcd(a, b) = gcd(b, r).

Note: While (2) and (3) are interesting exercises, Python comes with this functionality: check out the pow and math.gcd functions. The pow function can also be called using an operator: pow(4, 3) == 4**3. NOTE: 4**3 != 4^3. See Project 5 Challenges for more about this.

(4) Rewrite binary search (project 8-1) as a recursive function. Test it using test_isMember (project 8-2).