Lista de Exercícios 1 (Slides: Python 1)

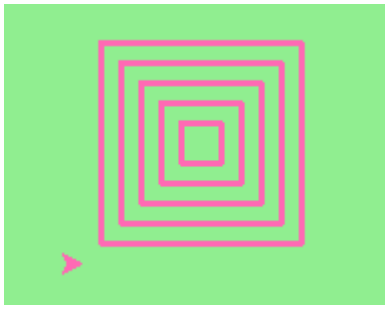
Instruções:

a) Os exercícios são individuais e devem ser entregues como projetos no GitHub.

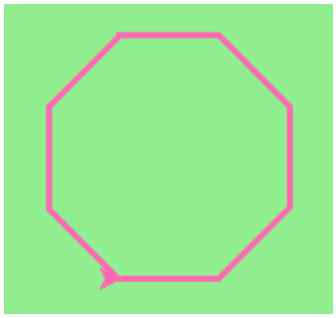
b) Enviar o link do site no GitHub através do Google Class.

c) Data final para entrega: 7/04/2020

1) Write a program to draw this. Assume the innermost square is 20 units per side, and each successive square is 20 units bigger, per side, than the one inside it.



2) Write a void function `draw_poly(t, n, sz)` which makes a turtle draw a regular polygon. When called with `draw_poly(tess, 8, 50)`, it will draw a shape like this:



3) Write a fruitful function `sum_to(n)` that returns the sum of all integer numbers up to and including `n`. So `sum_to(10)` would be *`1+2+3...+10`* which would return the value 55.

4) Sum all the elements in a list up to but not including the first even number. (Write your unit tests. What if there is no even number?)

5) Count how many words occur in a list up to and including the first occurrence of the word “sam”. (Write your unit tests for this case too. What if “sam” does not occur?)

6) Write a function, `is_prime`, which takes a single integer argument and returns `True` when the argument is a *prime number* and `False` otherwise. Add tests for cases like this:

```
test(is_prime(11))
test(not is_prime(35))
test(is_prime(19911121))
```

The last case could represent your birth date. Were you born on a prime day? In a class of 100 students, how many do you think would have prime birth dates?

7) Write a function `sum_of_squares(xs)` that computes the sum of the squares of the numbers in the list `xs`. For example, `sum_of_squares([2, 3, 4])` should return 4+9+16 which is 29:

```
test(sum_of_squares([2, 3, 4]) == 29)
test(sum_of_squares([ ]) == 0)
test(sum_of_squares([2, -3, 4]) == 29)
```

8) Print a neat looking multiplication table like this:

```
       1   2   3   4   5   6   7   8   9  10  11  12
  :------------------------------------------------
 1:     1   2   3   4   5   6   7   8   9  10  11  12
 2:     2   4   6   8  10  12  14  16  18  20  22  24
 3:     3   6   9  12  15  18  21  24  27  30  33  36
 4:     4   8  12  16  20  24  28  32  36  40  44  48
 5:     5  10  15  20  25  30  35  40  45  50  55  60
 6:     6  12  18  24  30  36  42  48  54  60  66  72
 7:     7  14  21  28  35  42  49  56  63  70  77  84
 8:     8  16  24  32  40  48  56  64  72  80  88  96
 9:     9  18  27  36  45  54  63  72  81  90  99 108
10:    10  20  30  40  50  60  70  80  90 100 110 120
11:    11  22  33  44  55  66  77  88  99 110 121 132
12:    12  24  36  48  60  72  84  96 108 120 132 144
```

9) Write a function that recognizes palindromes. (Hint: use your `reverse` function to make this easy!):

```
1
2 test(is_palindrome("abba"))
3 test(not is_palindrome("abab"))
4 test(is_palindrome("tenet"))
```

```
test(not is_palindrome("banana"))
test(is_palindrome("straw warts"))
test(is_palindrome("a"))
# test(is_palindrome(""))    # Is an empty string a palindrome?
```

10) A complex number is a number of the form a + bi, whereby a and b are constants, and i is a special value that is defined as the square root of -1. Of course, you never try to actually calculate what the square root of -1 is, as that gives a runtime error; in complex numbers, you always let the i remain. For instance, the complex number 3 + 2i cannot be simplified any further. Addition of two complex numbers a + bi and c + di is defined as (a + c) + (b + d)i. Represent a complex number as a tuple of two numeric values, and create a function that calculates the addition of two complex numbers.