Aula 2

Funções, strings, tuplas e listas

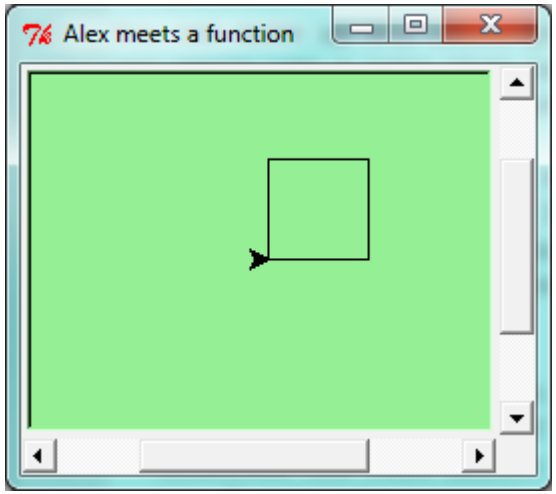
Objetivos

- Funções
- Strings
- Tuplas e listas

Função

```
import turtle
def  draw_square(t, sz):
  """Make turtle t draw a square of sz."""
    for i in range(4):
       t.forward(sz)
         t.left(90)

wn = turtle.Screen() # Set up the window and its attributes
wn.bgcolor("lightgreen")
wn.title("Alex meets a function")
alex = turtle.Turtle()     # Create alex
draw_square(alex, 50)  # Call the function to draw the square
wn.mainloop()
```


Alex meets a function

Funções retornam valores

$$A = P\left(1 + \frac{r}{n}\right)^{nt}$$

Where,

- P = principal amount (initial investment)
- r = annual nominal interest rate (as a decimal)
- n = number of times the interest is compounded per year
- t = number of years

```
1  def final_amt(p, r, n, t):
2      """
3        Apply the compound interest formula to p
4         to produce the final amount.
5      """
6
7      a = p * (1 + r/n) ** (n*t)
8      return a          # This is new, and makes the function fruitful.
9
10 # now that we have the function above, let us call it.
11 toInvest = float(input("How much do you want to invest?"))
12 fnl = final_amt(toInvest, 0.08, 12, 5)
13 print("At the end of the period you'll have", fnl)
```

Variaveis e parâmetros são locais

```
1  def final_amt(p, r, n, t):
2      a = p * (1 + r/n) ** (n*t)
3      return a
```

```
>>> a
NameError: name 'a' is not defined
```

```
1  def make_window(colr, ttle):
2      """
3        Set up the window with the given background color and title.
4        Returns the new window.
5      """
6      w = turtle.Screen()
7      w.bgcolor(colr)
8      w.title(ttle)
9      return w
10
11
12 def make_turtle(colr, sz):
13     """
14       Set up a turtle with the given color and pensize.
15       Returns the new turtle.
16     """
17     t = turtle.Turtle()
18     t.color(colr)
19     t.pensize(sz)
20     return t
21
22
23 wn = make_window("lightgreen", "Tess and Alex dancing")
24 tess = make_turtle("hotpink", 5)
25 alex = make_turtle("black", 1)
26 dave = make_turtle("yellow", 2)
```

Exercícios

- 4.9.9 e 4.9.10

Condicionais

```
>>> 5 == (3 + 2)   # Is five equal 5 to the result of 3 + 2?
True
>>> 5 == 6
False
>>> j = "hel"
>>> j + "lo" == "hello"
True
```

```
x == y              # Produce True if ... x is equal to y
x != y              # ... x is not equal to y
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 less than or equal to y
```

Execução condicional

```
1  if x % 2 == 0:
2      print(x, " is even.")
3      print("Did you know that 2 is the only even number that is prime?")
4  else:
5      print(x, " is odd.")
6      print("Did you know that multiplying two odd numbers " +
7                                                  "always gives an odd result?")
```

```
1  if x < 0:
2      print("The negative number ",  x, " is not valid here.")
3      x = 42
4      print("I've decided to use the number 42 instead.")
5
6  print("The square root of ", x, "is", math.sqrt(x))
```

Cont.

```
1  if choice == "a":
2      function_one()
3  elif choice == "b":
4      function_two()
5  elif choice == "c":
6      function_three()
7  else:
8      print("Invalid choice.")
```

```
1  if 0 < x and x < 10:
2      print("x is a positive single digit.")
```

Programação com estilo

- Usar 4 espaços (ao invés de tabs) para identação.
- Limitar o comprimento da linha em até 78 caracteres.
- Para nomes de identificadores:
  - Usar CamelCase para classes.
  - Usar lowercase_with_underscores para funções e variáveis.
- Colocar imports no topo de cada arquivo.
- Manter as definições de funções próximas.
- Usar 2 linhas brancas para separar funções.
- Usar docstrings para documentar funções.
- Manter o código com as chamadas para funções na parte baixa do programa.

Teste de Unidades

- Testar é uma atividade muitas vezes ignorada pelos alunos e programadores amadores.
- Uma boa prática é incluir testes junto com o código desenvolvido.
- TDD (Test Driven Development) é uma metodologia de desenvolvimento ágil onde os programadores desenvolvem testes antes do código.

```
1   import sys
2
3   def test(did_pass):
4       """  Print the result of a test.  """
5       linenum = sys._getframe(1).f_lineno   # Get the caller's line number.
6       if did_pass:
7           msg = "Test at line {0} ok.".format(linenum)
8       else:
9           msg = ("Test at line {0} FAILED.".format(linenum))
10      print(msg)
```

```
1  def absolute_value(x):
2      if x < 0:
3          return -x
4      return x
```

```
def test_suite():
    """ Run the suite of tests for code in this module (this file).
    """
    test(absolute_value(17) == 17)
    test(absolute_value(-17) == 17)
    test(absolute_value(0) == 0)
    test(absolute_value(3.14) == 3.14)
    test(absolute_value(-3.14) == 3.14)

test_suite()        # Here is the call to run the tests
```

```
Test at line 25 ok.
Test at line 26 ok.
Test at line 27 ok.
Test at line 28 ok.
Test at line 29 ok.
```

```
1  def absolute_value(n):   # Buggy version
2      """ Compute the absolute value of n """
3      if n < 0:
4          return 1
5      elif n > 0:
6          return n
```

```
Test at line 25 ok.
Test at line 26 FAILED.
Test at line 27 FAILED.
Test at line 28 ok.
Test at line 29 FAILED.
```

Exercícios

- 6.9.16 e 6.9.17

Tabelas

```
1  for x in range(13):   # Generate numbers 0 to 12
2      print(x, "\t", 2**x)
```

```
0       1
1       2
2       4
3       8
4       16
5       32
6       64
7       128
8       256
9       512
10      1024
11      2048
12      4096
```

```
1  for i in range(1, 7):
2      print(2 * i, end="    ")
3  print()
```

```
2       4       6       8       10      12
```

```
def print_multiples(n):
    for i in range(1, 7):
        print(n * i, end="    ")
    print()
```

```
for i in range(1, 7):
    print_multiples(i)
```

```
1       2       3       4       5       6
2       4       6       8       10      12
3       6       9       12      15      18
4       8       12      16      20      24
5       10      15      20      25      30
6       12      18      24      30      36
```

```
1  def print_mult_table(high):
2      for i in range(1, high+1):
3          print_multiples(i)
```

```
1       2       3       4       5       6
2       4       6       8       10      12
3       6       9       12      15      18
4       8       12      16      20      24
5       10      15      20      25      30
6       12      18      24      30      36
7       14      21      28      35      42
```

```
1  def print_multiples(n, high):
2      for i in range(1, high+1):
3          print(n * i, end="   ")
4      print()
5
6  def print_mult_table(high):
7      for i in range(1, high+1):
8          print_multiples(i, high)
```

```
1       2       3       4       5       6       7
2       4       6       8       10      12      14
3       6       9       12      15      18      21
4       8       12      16      20      24      28
5       10      15      20      25      30      35
6       12      18      24      30      36      42
7       14      21      28      35      42      49
```

1						
2	4					
3	6	9				
4	8	12	16			
5	10	15	20	25		
6	12	18	24	30	36	
7	14	21	28	35	42	49

Pares de dados (Tuplas)

```
1  year_born = ("Paris Hilton", 1981)
```

```
1  celebs = [("Brad Pitt", 1963), ("Jack Nicholson", 1937),
2                                 ("Justin Bieber", 1994)]
```

```
1  print(celebs)
2  print(len(celebs))
```

```
[("Brad Pitt", 1963), ("Jack Nicholson", 1937), ("Justin Bieber", 1994)]
3
```

```
1   for (nm, yr) in celebs:
2       if yr < 1980:
3           print(nm)
```

```
Brad Pitt
Jack Nicholson
```

```
1  students = [
2      ("John", ["CompSci", "Physics"]),
3      ("Vusi", ["Maths", "CompSci", "Stats"]),
4      ("Jess", ["CompSci", "Accounting", "Economics", "Management"]),
5      ("Sarah", ["InfSys", "Accounting", "Economics", "CommLaw"]),
6      ("Zuki", ["Sociology", "Economics", "Law", "Stats", "Music"])]
```

```
1  # Print all students with a count of their courses.
2  for (name, subjects) in students:
3      print(name, "takes", len(subjects), "courses")
```

```
John takes 2 courses
Vusi takes 3 courses
Jess takes 4 courses
Sarah takes 4 courses
Zuki takes 5 courses
```

```
1  # Count how many students are taking CompSci
2  counter = 0
3  for (name, subjects) in students:
4      for s in subjects:                    # A nested loop!
5          if s == "CompSci":
6              counter += 1
7
8  print("The number of students taking CompSci is", counter)
```

```
The number of students taking CompSci is 3
```

Exercícios

- 7.26.12 e 7.26.13

Strings

- String é um tipo (classe) primitivo composto tal como listas e as tuplas.
- Como um objeto de uma classe ele atende o conjunto de funções (métodos) da classe.

```
>>> ss = "Hello, World!"
>>> tt = ss.upper()
>>> tt
'HELLO, WORLD!'
```

```
>>> fruit = "banana"
>>> m = fruit[0]
>>> print(m)
b
```

```
>>> fruit = "banana"
>>> len(fruit)
6
```

```
sz = len(fruit)
last = fruit[sz]        # ERROR!
```

```
sz = len(fruit)
last = fruit[sz-1]
```

```
1  prefixes = "JKLMNOPQ"
2  suffix = "ack"
3
4  for p in prefixes:
5      print(p + suffix)
```

```
Jack
Kack
Lack
Mack
Nack
Oack
Pack
Qack
```

Fatias

```
>>> s = "Pirates of the Caribbean"
>>> print(s[0:7])
Pirates
>>> print(s[11:14])
the
>>> print(s[15:24])
Caribbean
>>> friends = ["Joe", "Zoe", "Brad", "Angelina", "Zuki", "Thandi", "Paris"]
>>> print(friends[2:4])
['Brad', 'Angelina']
```

```
>>> fruit = "banana"
>>> fruit[:3]
'ban'
>>> fruit[3:]
'ana'
>>> fruit[3:999]
'ana'
```

Comparação de Strings

```
1  if word == "banana":
2      print("Yes, we have no bananas!")
```

```
1  if word < "banana":
2      print("Your word, " + word + ", comes before banana.")
3  elif word > "banana":
4      print("Your word, " + word + ", comes after banana.")
5  else:
6      print("Yes, we have no bananas!")
```

```
Your word, Zebra, comes before banana.
```

Strings são imutáveis

```
1  greeting = "Hello, world!"
2  greeting[0] = 'J'              # ERROR!
3  print(greeting)
```

```
1  greeting = "Hello, world!"
2  new_greeting = "J" + greeting[1:]
3  print(new_greeting)
```

Operadores **in** e **+** (concatenação)

```
1   def remove_vowels(s):
2       vowels = "aeiouAEIOU"
3       s_sans_vowels = ""
4       for x in s:
5           if x not in vowels:
6               s_sans_vowels += x
7       return s_sans_vowels
8
9   test(remove_vowels("compsci") == "cmpsc")
10  test(remove_vowels("aAbEefIijOopUus") == "bfjps")
```

Método format

```
1   s1 = "His name is {0}!".format("Arthur")
2   print(s1)
3
4   name = "Alice"
5   age = 10
6   s2 = "I am {1} and I am {0} years old.".format(age, name)
7   print(s2)
8
9   n1 = 4
10  n2 = 5
11  s3 = "2**10 = {0} and {1} * {2} = {3:f}".format(2**10, n1, n2, n1 * n2)
12  print(s3)
```

```
His name is Arthur!
I am Alice and I am 10 years old.
2**10 = 1024 and 4 * 5 = 20.000000
```

Exercícios

- 8.19.10

Tuplas

- Tuplas são agrupamentos de dados imutáveis.

```
>>> year_born = ("Paris Hilton", 1981)
```

```
>>> julia = ("Julia", "Roberts", 1967, "Duplicity", 2009, "Actress", "Atlanta, Geo
```

```
>>> julia[2]
1967
```

```
>>> julia[0] = "X"
TypeError: 'tuple' object does not support item assignment
```

```
>>> tup = (5,)
>>> type(tup)
<class 'tuple'>
>>> x = (5)
>>> type(x)
<class 'int'>
```

```
>>> b = ("Bob", 19, "CS")
>>> (name, age, studies) = b    # tuple unpacking
>>> name
'Bob'
>>> age
19
>>> studies
'CS'
```

```
1  (a, b) = (b, a)
```

```
def f(r):
    """ Return (circumference, area) of a circle of radius r """
    c = 2 * math.pi * r
    a = math.pi * r * r
    return (c, a)
```

```
julia_more_info = ( ("Julia", "Roberts"), (8, "October", 1967),
                    "Actress", ("Atlanta", "Georgia"),
                    [ ("Duplicity", 2009),
                      ("Notting Hill", 1999),
                      ("Pretty Woman", 1990),
                      ("Erin Brockovich", 2000),
                      ("Eat Pray Love", 2010),
                      ("Mona Lisa Smile", 2003),
                      ("Oceans Twelve", 2004) ])
```

Listas

- Listas são coleções ordenadas de objetos.

```
ps = [10, 20, 30, 40]
qs = ["spam", "bungee", "swallow"]
```

```
zs = ["hello", 2.0, 5, [10, 20]]
```

```
1  >>> vocabulary = ["apple", "cheese", "dog"]
2  >>> numbers = [17, 123]
3  >>> an_empty_list = []
4  >>> print(vocabulary, numbers, an_empty_list)
5  ["apple", "cheese", "dog"] [17, 123] []
```

```
1  horsemen = ["war", "famine", "pestilence", "death"]
2
3  for h in horsemen:
4      print(h)
```

```
1  horsemen = ["war", "famine", "pestilence", "death"]
2
3  for i in range(len(horsemen)):
4      print(horsemen[i])
```

```
>>> len(["car makers", 1, ["Ford", "Toyota", "BMW"], [1, 2, 3]])
4
```

```
>>> horsemen = ["war", "famine", "pestilence", "death"]
>>> "pestilence" in horsemen
True
>>> "debauchery" in horsemen
False
>>> "debauchery" not in horsemen
True
```

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> c
[1, 2, 3, 4, 5, 6]
```

```
>>> [0] * 4
[0, 0, 0, 0]
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
>>> a_list = ["a", "b", "c", "d", "e", "f"]
>>> a_list[1:3]
['b', 'c']
>>> a_list[:4]
['a', 'b', 'c', 'd']
>>> a_list[3:]
['d', 'e', 'f']
>>> a_list[:]
['a', 'b', 'c', 'd', 'e', 'f']
```

```
>>> fruit = ["banana", "apple", "quince"]
>>> fruit[0] = "pear"
>>> fruit[2] = "orange"
>>> fruit
['pear', 'apple', 'orange']
```

```
>>> a_list = ["a", "b", "c", "d", "e", "f"]
>>> a_list[1:3] = ["x", "y"]
>>> a_list
['a', 'x', 'y', 'd', 'e', 'f']
```

```
>>> a_list = ["a", "b", "c", "d", "e", "f"]
>>> a_list[1:3] = []
>>> a_list
['a', 'd', 'e', 'f']
```

```
>>> a_list = ["a", "d", "f"]
>>> a_list[1:1] = ["b", "c"]
>>> a_list
['a', 'b', 'c', 'd', 'f']
>>> a_list[4:4] = ["e"]
>>> a_list
['a', 'b', 'c', 'd', 'e', 'f']
```

```
>>> a = ["one", "two", "three"]
>>> del a[1]
>>> a
['one', 'three']
```

```
>>> a_list = ["a", "b", "c", "d", "e", "f"]
>>> del a_list[1:5]
>>> a_list
['a', 'f']
```

Objetos e referencias

```
>>> a = [1, 2, 3]
>>> b = [1, 2, 3]
>>> a == b
True
>>> a is b
False
```

```
>>> a = [1, 2, 3]
>>> b = a
>>> a is b
True
```

Clonar lista

```
>>> a = [1, 2, 3]
>>> b = a[:]

>>> b
[1, 2, 3]
```

Listas e for loops

```
1  friends = ["Joe", "Zoe", "Brad", "Angelina", "Zuki", "Thandi", "Paris"]
2  for friend in friends:
3      print(friend)
```

```
1  xs = [1, 2, 3, 4, 5]
2
3  for i in range(len(xs)):
4      xs[i] = xs[i]**2
```

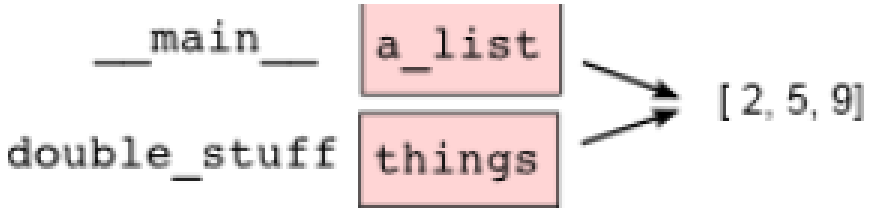
```
1  for (i, v) in enumerate(["banana", "apple", "pear", "lemon"]):
2       print(i, v)
```

```
0 banana
1 apple
2 pear
3 lemon
```

Lista como parâmetro

```
1  def double_stuff(a_list):
2      """ Overwrite each element in a_list with double its value. """
3      for (idx, val) in enumerate(a_list):
4          a_list[idx] = 2 * val
```

```
1  things = [2, 5, 9]
2  double_stuff(things)
3  print(things)
```




```
[4, 10, 18]
```

Métodos de lista

```
>>> mylist = []
>>> mylist.append(5)
>>> mylist.append(27)
>>> mylist.append(3)
>>> mylist.append(12)
>>> mylist
[5, 27, 3, 12]
```

Cont.

```
>>> mylist.insert(1, 12)  # Insert 12 at pos 1, shift other items up
>>> mylist
[5, 12, 27, 3, 12]
>>> mylist.count(12)       # How many times is 12 in mylist?
2
>>> mylist.extend([5, 9, 5, 11])   # Put whole list onto end of mylist
>>> mylist
[5, 12, 27, 3, 12, 5, 9, 5, 11])
>>> mylist.index(9)                # Find index of first 9 in mylist
6
>>> mylist.reverse()
>>> mylist
[11, 5, 9, 5, 12, 3, 27, 12, 5]
>>> mylist.sort()
>>> mylist
[3, 5, 5, 5, 9, 11, 12, 12, 27]
>>> mylist.remove(12)             # Remove the first 12 in the list
>>> mylist
[3, 5, 5, 5, 9, 11, 12, 27]
```

Funções Puras

- Uma função pura não produz efeitos colaterais. A comunicação é somente através dos parâmetros e do valor retornado.

```
1   def double_stuff(a_list):
2       """ Return a new list which contains
3           doubles of the elements in a_list.
4       """
5       new_list = []
6       for value in a_list:
7           new_elem = 2 * value
8           new_list.append(new_elem)
9
10      return new_list
```

```
>>> things = [2, 5, 9]
>>> xs = double_stuff(things)
>>> things
[2, 5, 9]
>>> xs
[4, 10, 18]
```

Strings e listas

```
>>> song = "The rain in Spain..."
>>> wds = song.split()
>>> wds
['The', 'rain', 'in', 'Spain...']
```

```
>>> song.split("ai")
['The r', 'n in Sp', 'n...']
```

```
>>> glue = ";"
>>> s = glue.join(wds)
>>> s
'The;rain;in;Spain...'
```

funções lista e range

```
>>> xs = list("Crunchy Frog")
>>> xs
["C", "r", "u", "n", "c", "h", "y", " ", "F", "r", "o", "g"]
>>> "".join(xs)
'Crunchy Frog'
```

```
1   def f(n):
2       """ Find the first positive integer between 101 and less
3           than n that is divisible by 21
4       """
5       for i in range(101, n):
6          if (i % 21 == 0):
7              return i
8
9
10  test(f(110) == 105)
11  test(f(1000000000) == 105)
```

Lista aninhada

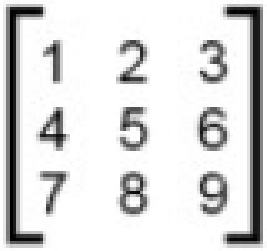
```
>>> nested = ["hello", 2.0, 5, [10, 20]]
```

```
>>> print(nested[3])
[10, 20]
```

```
>>> elem = nested[3]
>>> elem[0]
10
```

```
>>> nested[3][1]
20
```

Matrizes




```
>>> mx = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
>>> mx[1]
[4, 5, 6]
```

```
>>> mx[1][2]
6
```

Exercícios

- 11.22.5 e 11.22.6