CES 22 - 2018

Aula 3

Objetivos

- Programação dirigida por eventos
- Módulos
- Objetos e Classes

Programação dirigida por Eventos

- A maioria dos programas com interfaces gráficas respondem a eventos. Por exemplo, ao clickar um botão, uma nova janela se abre.
- Para cada tipo de evento está registrado uma ou mais funções (ou rotinas) que são ativadas quando o evento acontece.

```python
import turtle

turtle.setup(400,500)                   # Determine the window size
wn = turtle.Screen()                    # Get a reference to the window
wn.title("Handling keypresses!")        # Change the window title
wn.bgcolor("lightgreen")                # Set the background color
tess = turtle.Turtle()                  # Create our favorite turtle

# The next four functions are our "event handlers".
def h1():
   tess.forward(30)

def h2():
   tess.left(45)

def h3():
   tess.right(45)

def h4():
    wn.bye()                           # Close down the turtle window

# These lines "wire up" keypresses to the handlers we've defined.
wn.onkey(h1, "Up")
wn.onkey(h2, "Left")
wn.onkey(h3, "Right")
wn.onkey(h4, "q")

# Now we need to tell the window to start listening for events,
# If any of the keys that we're monitoring is pressed, its
# handler will be called.
wn.listen()
wn.mainloop()
```

```python
import turtle

turtle.setup(400,500)
wn = turtle.Screen()
wn.title("How to handle mouse clicks on the window!")
wn.bgcolor("lightgreen")

tess = turtle.Turtle()
tess.color("purple")
tess.pensize(3)
tess.shape("circle")

def h1(x, y):
    tess.goto(x, y)

wn.onclick(h1)  # Wire up a click on the window.
wn.mainloop()
```

```python
import turtle

turtle.setup(400,500)
wn = turtle.Screen()
wn.title("Using a timer")
wn.bgcolor("lightgreen")

tess = turtle.Turtle()
tess.color("purple")
tess.pensize(3)

def h1():
    tess.forward(100)
    tess.left(56)

wn.ontimer(h1, 2000)
wn.mainloop()
```

```python
import turtle

turtle.setup(400,500)
wn = turtle.Screen()
wn.title("Using a timer to get events!")
wn.bgcolor("lightgreen")

tess = turtle.Turtle()
tess.color("purple")

def h1():
    tess.forward(100)
    tess.left(56)
    wn.ontimer(h1, 60)

h1()
wn.mainloop()
```

Exercício

- 10.6.2

Módulos

- Módulo é um arquivo contendo definições e declarações que podem ser utilizadas em outros programas Python. Exemplos: módulo turtle e string.
- help(*nomeDoModulo*)

```
import time

def do_my_sum(xs):
    sum = 0
    for v in xs:
        sum += v
    return sum

sz = 10000000        # Lets have 10 million elements in the list
testdata = range(sz)

t0 = time.clock()
my_result = do_my_sum(testdata)
t1 = time.clock()
print("my_result    = {0} (time taken = {1:.4f} seconds)"
        .format(my_result, t1-t0))

t2 = time.clock()
their_result = sum(testdata)
t3 = time.clock()
print("their_result = {0} (time taken = {1:.4f} seconds)"
        .format(their_result, t3-t2))
```

```
my_sum    = 49999995000000 (time taken = 1.5567 seconds)
their_sum = 49999995000000 (time taken = 0.9897 seconds)
```

```
>>> import math
>>> math.pi                # Constant pi
3.141592653589793
>>> math.e                 # Constant natural log base
2.718281828459045
>>> math.sqrt(2.0)         # Square root function
1.4142135623730951
>>> math.radians(90)       # Convert 90 degrees to radians
1.5707963267948966
>>> math.sin(math.radians(90))  # Find sin of 90 degrees
1.0
>>> math.asin(1.0) * 2     # Double the arcsin of 1.0 to get pi
3.141592653589793
```

Criar módulos

- Salvar o script em um arquivo com extensão .py
- Exemplo: criar um módulo seqtools.py

```python
def remove_at(pos, seq):
    return seq[:pos] + seq[pos+1:]
```

```python
>>> import seqtools
>>> s = "A string!"
>>> seqtools.remove_at(4, s)
'A sting!'
```

Namespace

- Coleção de identificadores que pertencem a módulos, funções e classes. Evita colisão de nomes.

```python
# Module1.py

question = "What is the meaning of Life, the Universe, and Everything?"
answer = 42
```

```python
# Module2.py

question = "What is your quest?"
answer = "To seek the holy grail."
```

```python
import module1
import module2

print(module1.question)
print(module2.question)
print(module1.answer)
print(module2.answer)
```

Escopo de variaveis

- Escopo local: identificadores declarados dentro de uma função.
- Escopo Global: identificadores declarados dentro de um módulo ou arquivo.
- Escopo Built-in: identificadores built-in de Python.

```
def range(n):
    return 123*n

print(range(10))
```

O que será impresso?

```
n = 10
m = 3
def f(n):
    m = 7
    return 2*n+m

print(f(5), n, m)
```

This prints 17 10 3

Variantes de import

```python
import math
x = math.sqrt(10)
```

```python
from math import cos, sin, sqrt
x = sqrt(10)
```

```python
from math import *    # Import all the identifiers from math,
                      #   adding them to the current namespace.
x = sqrt(10)          # Use them without qualification.
```

```python
def area(radius):
    import math
    return math.pi * radius * radius

x = math.sqrt(10)      # This gives an error
```

Exercícios

- 12.11.8
- 13.11.1
- 14.11.1
- 14.11.2

Classes e Objetos

```python
class Point:
    """ Point class represents and manipulates x,y coords. """

    def __init__(self):
        """ Create a new point at the origin """
        self.x = 0
        self.y = 0
```

```python
p = Point()          # Instantiate an object of type Point
q = Point()          # Make a second point

print(p.x, p.y, q.x, q.y)  # Each point object has its own x and y
```

```
0 0 0 0
```

```python
class Point:
    """ Point class represents and manipulates x,y coords. """

    def __init__(self, x=0, y=0):
        """ Create a new point at x, y """
        self.x = x
        self.y = y

# Other statements outside the class continue below here.
```

```python
>>> p = Point(4, 2)
>>> q = Point(6, 3)
>>> r = Point()       # r represents the origin (0, 0)
>>> print(p.x, q.y, r.x)
4 3 0
```

```python
class Point:
    """ Create a new Point, at coordinates x, y """

    def __init__(self, x=0, y=0):
        """ Create a new point at x, y """
        self.x = x
        self.y = y

    def distance_from_origin(self):
        """ Compute my distance from the origin """
        return ((self.x ** 2) + (self.y ** 2)) ** 0.5
```

```
>>> p = Point(3, 4)
>>> p.x
3
>>> p.y
4
>>> p.distance_from_origin()
5.0
>>> q = Point(5, 12)
>>> q.x
5
>>> q.y
12
>>> q.distance_from_origin()
13.0
>>> r = Point()
>>> r.x
0
>>> r.y
0
>>> r.distance_from_origin()
0.0
```

```python
def print_point(pt):
    print("({0}, {1})".format(pt.x, pt.y))
```

```python
class Point:
    # ...

    def to_string(self):
        return "({0}, {1})".format(self.x, self.y)
```

```python
>>> p = Point(3, 4)
>>> print(p.to_string())
(3, 4)
```

```
>>> str(p)
'<__main__.Point object at 0x01F9AA10>'
>>> print(p)
'<__main__.Point object at 0x01F9AA10>'
```

```
class Point:
    # ...

    def __str__(self):    # All we have done is renamed the method
        return "({0}, {1})".format(self.x, self.y)
```

```
>>> str(p)     # Python now uses the __str__ method that we wrote.
(3, 4)
>>> print(p)
(3, 4)
```

```python
def midpoint(p1, p2):
    """ Return the midpoint of points p1 and p2 """
    mx = (p1.x + p2.x)/2
    my = (p1.y + p2.y)/2
    return Point(mx, my)
```

```python
class Point:
  # ...

  def halfway(self, target):
      """ Return the halfway point between myself and the target """
      mx = (self.x + target.x)/2
      my = (self.y + target.y)/2
      return Point(mx, my)
```

```
>>> p = Point(3, 4)
>>> q = Point(5, 12)
>>> r = p.halfway(q)
>>> r
(4.0, 8.0)
```

Exercícios

- 15.12.3
- 15.12.4