

Python Training

Sessions by Tiago Montes

Kids on Python | Guia de Experiências

Programas Python para experimentares

Versão 1.0.1, Agosto de 2019

Boas vindas!

Depois da tua participação nas sessões **Kids on Python** podes ter interesse em continuar a aprender.

Nesse caso, tens neste documento um conjunto de programas que podes ler, copiar para o Mu, executar e, claro, experimentar alterar para obteres resultados diferentes.

Diverte-te!

Experiências

1 Pistas Iniciais.....	3
2 Tartaruga O quadrado.....	4
3 Tartaruga Estrelas.....	5
4 Tartaruga Cores.....	6
5 Tartaruga Arte abstracta.....	7
6 Tartaruga Escrever texto.....	8
7 Tartaruga Desenhar com o rato.....	9
8 Tartaruga Jogo da Corrida.....	10
9 Tartaruga Jogo da Captura.....	13
10 Texto Cumprimentos.....	17
11 Texto Pedra, Papel, ou Tesoura?.....	18
12 Números Fazer contas.....	19
13 Números Ciclos de contagem.....	20
14 Números e Tartaruga Desenhar uma grelha.....	21
15 Números Adivinhar um Número.....	22
Parabéns!.....	23

1 | Pistas Iniciais

Aqui estão algumas pistas que te ajudam a explorar este guia:

- Cada programa é apresentado num capítulo diferente:
 - Em cada um deles estão sempre incluídas algumas explicações e pistas.
 - Ou nos comentários do programa, ou no texto associado.
- O código Python de cada programa está sempre dentro de uma caixa de fundo cinzento.
 - Nessa caixa é indicado o número de cada linha à esquerda, tal como no Mu.
 - Esses números servem só para te ajudar, já sabes que não fazem parte do programa!
- Às vezes podes deparar-te com erros que impedem o programa de funcionar.
 - Utiliza o botão “**Check**” / “**Verificar**” na barra superior do Mu, à medida que vais escrevendo o programa.
 - Muitas vezes pode ajudar-te a identificar algo que não está bem escrito.
 - Por exemplo, podes ter-te esquecido de fechar parêntesis, ou de colocar uma vírgula onde faça falta!

2 | Tartaruga | O quadrado

Já experimentaste este programa durante as sessões, mas aqui fica esta variante, com alguns comentários, onde o **TAMANHO**, **COR**, e **ESPESSURA** do quadrado são definidos com variáveis.

Inclui também, entre as linhas 4 e 15, uma receita que podes utilizar em qualquer programa que utilize o módulo **turtle**, para definir o tamanho e posição inicial da janela da tartaruga no ecrã.

```
1  import turtle
2
3
4  # Dimensões da Janela.
5  LARGURA = 400
6  ALTURA = 400
7
8  # Posição horizontal da janela: quanto maior, mais para o lado direito.
9  X = 800
10
11 # Posição vertical da janela: quanto maior, mais para baixo.
12 Y = 100
13
14 # Definir dimensão e posição da janela da tartaruga.
15 turtle.Screen().setup(width=LARGURA, height=ALTURA, startx=X, starty=Y)
16
17
18 # Variáveis com informação do quadrado.
19 TAMANHO = 150
20 COR = "blue"
21 ESPESSURA = 5
22
23 # Construir uma tartaruga.
24 t = turtle.Turtle()
25
26 # Definir a cor e espessura da linha.
27 t.color(COR)
28 t.width(ESPESSURA)
29
30 # Desenhar o quadrado.
31 t.forward(TAMANHO)
32 t.left(90)
33 t.forward(TAMANHO)
34 t.left(90)
35 t.forward(TAMANHO)
36 t.left(90)
37 t.forward(TAMANHO)
```

3 | Tartaruga | Estrelas

Estrelas (ou coisas muito parecidas!) com a repetição das instruções **.forward** e **.right** de uma tartaruga.

Pista: Muitos destes programas não incluem as instruções para definir a posição e o tamanho da janela da tartaruga. Claro, se quiseses podes sempre alterá-los, com inspiração no programa anterior!

```
1  import turtle
2
3  # Experimenta alterar estes valores.
4  DISTANCIA = 200
5  ANGULO = 160
6  VEZES = 9
7
8  t = turtle.Turtle()
9  t.width(3)
10
11 # Repetir t.forward e t.right.
12 for vez in range(VEZES):
13     t.forward(DISTANCIA)
14     t.right(ANGULO)
```

Uma variante do programa, que vai desenhado quadrados, rodando um bocadinho de cada vez:

```
1  import turtle
2
3  # Experimenta alterar estes valores.
4  DISTANCIA = 200
5  ANGULO = 160
6  VEZES = 9
7
8  t = turtle.Turtle()
9  t.width(3)
10 t.speed(10)
11
12 def quadrado():
13     for vez in range(4):
14         t.forward(DISTANCIA)
15         t.left(90)
16
17 # Repetir quadrados, rodando sempre um bocadinho.
18 for vez in range(VEZES):
19     quadrado()
20     t.right(ANGULO)
```

4 | Tartaruga | Cores

Quando se usa o módulo **turtle** do Python há duas formas de definir as cores: ou com nomes em Inglês, entre aspas, ou com um conjunto de três números decimais, entre **0.0** e **1.0**, separados por vírgulas.

O programa seguinte, desenha várias linhas de cor diferente, seguindo ambas os métodos.

```
1  import turtle
2
3  # Tartaruga com linha grossa, virada para baixo.
4  t = turtle.Turtle()
5  t.width(40)
6  t.right(90)
7
8  # Uma função que desenha linhas verticais na posição x.
9  # (valores negativos à esquerda, positivos à direita)
10 def linha_vertical(x):
11     t.up()
12     t.goto(x, 100)
13     t.down()
14     t.forward(200)
15
16 t.color("black")
17 linha_vertical(-140)
18
19 t.color("blue")
20 linha_vertical(-100)
21
22 t.color(1.0, 0.0, 1.0)
23 linha_vertical(-60)
24
25 t.color("red")
26 linha_vertical(-20)
27
28 t.color("orange")
29 linha_vertical(20)
30
31 t.color("yellow")
32 linha_vertical(60)
33
34 t.color("green")
35 linha_vertical(100)
36
37 t.color("gray")
38 linha_vertical(140)
```

5 | Tartaruga | Arte abstracta

Este programa faz desenhos de arte abstracta com a repetição de linhas aleatórias (ao calhas, lembrás-te?) utilizando coordenadas cartesianas X e Y para movimentar a tartaruga.

Além disso, cada linha é desenhada com uma cor e espessura aleatórias.

```
1  import turtle
2  import random
3
4
5  QUANTAS_LINHAS = 50
6
7
8  # Uma tartaruga muito rápida a desenhar.
9  t = turtle.Turtle()
10 t.speed(0)
11
12 # Vamos repetir o desenho de uma linha.
13 for vez in range(QUANTAS_LINHAS):
14
15     # Cor da tartaruga aleatória.
16     r = random.randint(0, 10) / 10
17     g = random.randint(0, 10) / 10
18     b = random.randint(0, 10) / 10
19     t.color(r, g, b)
20
21     # Espessura aleatória.
22     espessura = random.randint(3, 10)
23     t.width(espessura)
24
25     # Destino da tartaruga aleatório.
26     x = random.randint(-200, 200)
27     y = random.randint(-200, 200)
28
29     # Desenhar a linha!
30     t.goto(x, y)
```

Lembra-te que a instrução **random.randint(mínimo, máximo)** representa um número inteiro aleatório, entre os valores *mínimo* e *máximo*.

Nota como o cálculo para as variáveis **r**, **g**, e **b**, que definem uma cor aleatória, é feito a partir de um número entre 0 e 10 com a expressão **random.randint(0, 10)**. Esse número é depois dividido por 10, com a expressão **/ 10** por forma a obter um resultado final entre **0.0** e **1.0** – tal como a instrução **.color** da tartaruga precisa.

6 | Tartaruga | Escrever texto

Ainda que não tenhamos visto com muita atenção durante as sessões, o módulo **turtle** do Python também pode, além de desenhar linhas, escrever texto.

Aqui fica um exemplo para explorares esta possibilidade.

```
1  import turtle
2
3  # A tartaruga para escrever texto em azul.
4  t = turtle.Turtle()
5  t.color("blue")
6
7  # Podemos esconder a tartaruga com a instrução .hideturtle()
8  t.hideturtle()
9
10 # Texto em cima, alinhado à esquerda: aparece à direita da tartaruga.
11 t.goto(0, 100)
12 t.write("Bom dia!", align="left", font=("Arial", 50, "bold"))
13
14 # Texto ao meio, alinhado ao centro: aparece centrado com a tartaruga.
15 t.goto(0, 0)
16 t.write("Boa tarde...", align="center", font=("Arial", 50, "bold"))
17
18 # Texto em baixo, alinhado à direita: aparece à esquerda da tartaruga.
19 t.goto(0, -100)
20 t.write("Boa noite.", align="right", font=("Arial", 50, "bold"))
```

Algumas notas acerca deste programa:

- A cor do texto é a cor da tartaruga, por isso todo o texto é azul.
Experimenta mudar o programa por forma a que cada linha de texto tenha uma cor diferente.
- O tamanho do texto é definido pelo número **50** que podes encontrar nas instruções **.write**.
Experimenta números diferentes para obteres tamanhos diferentes (pista: 10 é pequeno, 100 é enorme).
- A string **"Arial"** nas instruções **.write** indica o tipo de letra, algo que talvez já possas conhecer da utilização de programas de processamento de texto. Há imensos tipos de letra diferentes, mas podem variar muito entre computadores diferentes. Se quiseses experimentar é provável que alguns dos seguintes tipos funcionem: **"Times New Roman"**, **"Courier New"** ou **"Comic Sans MS"**.
- Por último, a string **"bold"** nas instruções **.write** indica a variante do tipo de letra.
Em alternativa podes experimentar **"italic"** ou **"normal"**.

7 | Tartaruga | Desenhar com o rato

Neste programa encontras uma receita para reagir a cliques com o cursor do rato, fazendo desenhos.

Como funciona: a tartaruga começa, como sempre, no centro geométrico da janela. Conforme utilizas o rato, fazendo clique em sítios diferentes, assim se move a tartaruga, desenhando uma linha! As teclas “u” e “d” levantam e baixam a tartaruga para que a possas mover sem desenhar uma linha, se quiseres.

```
1  import turtle
2
3
4  # A tartaruga para desenhar.
5  t = turtle.Turtle()
6  t.width(3)
7  t.color("magenta")
8  t.shape("circle")
9
10
11 def move_tartaruga(x, y):
12     # Move a tartaruga para a posição do rato!
13     t.goto(x, y)
14
15 def levanta_tartaruga():
16     # Levanta a tartaruga para não desenhar linhas: forma de quadrado.
17     t.shape("square")
18     t.up()
19
20 def baixa_tartaruga():
21     # Baixa a tartaruga para voltar a desenhar linhas: forma de circulo.
22     t.shape("circle")
23     t.down()
24
25
26 # Cliques do rato executam a função "move_tartaruga".
27 turtle.onscreenclick(move_tartaruga)
28
29 # Preparar para usar o teclado.
30 turtle.listen()
31
32 # As teclas "u" e "d" executam as respectivas funções.
33 turtle.onkey(levanta_tartaruga, "u")
34 turtle.onkey(baixa_tartaruga, "d")
35
36 # Aguarda que a janela feche ou que se utilize o rato e teclado.
37 turtle.mainloop()
```

8 | Tartaruga | Jogo da Corrida

Este é um programa bastante comprido que faz um jogo de corrida aleatória.

Como funciona:

No início aparecem dois aviões do lado esquerdo e a linha da meta do lado direito. Depois, de cada vez que se carrega na tecla “c”, ambos os aviões avançam de forma aleatória (ao calhas, lembras-te?). O primeiro que ultrapassar a meta é o vencedor. Nos casos em que na mesma jogada ambos a ultrapassam dá-se um empate!

Nota: Para que este programa funcione no teu computador tal como está escrito, os ficheiros GIF com as imagens dos aviões têm que estar presentes na pasta “**mu_code**” do teu computador. Se não tiveres as imagens dos ficheiros chamados “**aviao-a-1.gif**” e “**aviao-b-4.gif**”, experimenta comentar as instruções com referência às variáveis **IMAGEM_1** e **IMAGEM_2**. Se o fizeres, o programa funciona com os símbolos base das tartarugas.

```
1  import turtle
2  import random
3
4
5  # Preparar a janela do jogo.
6  turtle.Screen().setup(width=600, height=400, startx=840)
7  turtle.bgcolor("skyblue")
8
9
10 # Uma tartaruga para desenhar a meta.
11 t = turtle.Turtle()
12 t.speed(10)
13 t.width(5)
14 t.color("white")
15 t.up()
16 t.forward(200)
17 t.left(90)
18 t.down()
19 t.forward(200)
20 t.left(180)
21 t.forward(400)
22
23
24 # As imagens para as tartarugas corredoras.
25 # (estes ficheiros têm que existir na pasta "mu_code" do teu computador!)
26 IMAGEM_1 = "aviao-a-1.gif"
27 IMAGEM_2 = "aviao-b-4.gif"
28
29
30
31
```

```
32 # Uma tartaruga que é o corredor de cima.
33 cima = turtle.Turtle()
34 turtle.register_shape(IMAGEM_1)
35 cima.shape(IMAGEM_1)
36 cima.color("white")
37 cima.up()
38
39 # Vai para a partida, à esquerda em cima.
40 cima.left(180)
41 cima.forward(200)
42 cima.right(90)
43 cima.forward(100)
44 cima.right(90)
45
46
47 # Uma tartaruga que é o corredor de baixo.
48 baixo = turtle.Turtle()
49 turtle.register_shape(IMAGEM_2)
50 baixo.shape(IMAGEM_2)
51 baixo.color("white")
52 baixo.up()
53
54 # Vai para a partida, à esquerda em baixo.
55 baixo.left(180)
56 baixo.forward(200)
57 baixo.left(90)
58 baixo.forward(100)
59 baixo.left(90)
60
61
62 # A função que executa cada que se carregar numa tecla.
63 def corrida():
64
65     # Se o corredor de cima ultrapassou a meta, com coordenada X > 200,
66     # ou o corredor de baixo ultrapassou a meta, com coordenada X > 200,
67     # então o jogo terminou e instrução return interrompe esta função.
68     if cima.xcor() > 200 or baixo.xcor() > 200:
69         return
70
71     # Determinar passos aleatórios para cada corredor.
72     passos_cima = random.randint(50, 100)
73     passos_baixo = random.randint(50, 100)
74
75     # Cada corredor avança.
76     cima.forward(passos_cima)
77     baixo.forward(passos_baixo)
```

```
78
79     # Se ambos os corredores ultrapassaram a meta, com coordenada X > 200,
80     # é um empate: escreve a mensagem "Empate" junto a cada corredor e termina.
81     if cima.xcor() > 200 and baixo.xcor() > 200:
82         cima.write("Empate", align="right", font=("Arial", 80, "bold"))
83         baixo.write("Empate", align="right", font=("Arial", 80, "bold"))
84         return
85
86     # Se o jogador de cima ultrapassou a meta escreve a mensagem "Ganhou!".
87     if cima.xcor() > 200:
88         cima.write("Ganhou!", align="right", font=("Arial", 80, "bold"))
89
90     # Se o jogador de baixo ultrapassou a meta escreve a mensagem "Ganhou!".
91     if baixo.xcor() > 200:
92         baixo.write("Ganhou!", align="right", font=("Arial", 80, "bold"))
93
94
95     # Prepara o programa para usar o teclado.
96     turtle.listen()
97
98     # Receita: sempre que se carrega na tecla "c" a função "corrida" é executada.
99     turtle.onkey(corrida, "c")
100
101     # Aguarda pelo teclado ou que se feche a janela.
102     turtle.mainloop()
```

Coisas que podes experimentar:

- Alterar a cor de fundo e da meta, por exemplo.
- Experimentar outras imagens.

Nas sessões que fizemos, transferimos para o teu computador um conjunto de ficheiros com imagens de carros, aviões, naves e alguns bonecos.

Vê que ficheiros estão disponíveis na pasta “**mu_editor**” do teu computador (ou pede a alguém que te possa ajudar) e experimenta-os ao teu gosto.

Deves encontrar, pelo menos: “**carro-1.gif**”, “**carro-2.gif**”, “**carro-3.gif**”, “**carro-4.gif**”, “**carro-5.gif**”, “**carro-6.gif**”, “**carro-7.gif**”, “**nave-1.gif**”, “**nave-2.gif**”, “**nave-3.gif**”, “**nave-4.gif**”, “**nave-5.gif**”, e “**nave-6.gif**”.

9 | Tartaruga | Jogo da Captura

Este é um dos programas mais complexos neste guia.

Como no caso do Jogo da Corrida, para que funcione tal como está escrito, precisa que os ficheiros GIF com imagens dos jogadores estejam disponíveis na pasta “**mu_code**” do teu computador. Os ficheiros chamam-se “**boneco-c-animal.gif**” e “**boneco-c-cavaleiro.gif**” e deverás tê-los, se participaste nas nossas sessões.

Caso contrário, elimina as linhas com referências a esses ficheiros e o programa funcionará, mas apenas com os símbolos base das tartarugas, para representar os jogadores.

Como funciona:

Há um jogador e um fugitivo. Ambos se deslocam numa grelha pré-definida.

As teclas de direcção movem o jogador que deve tentar apanhar o fugitivo.

Por cada movimento o jogador perde um bocadinho de energia, por cada captura ganha bastante energia.

Quando a energia chegar ao mínimo o jogador perde, se chegar ao máximo o jogador ganha!

```
1  import turtle
2  import random
3
4
5  # Quanta energia gasta cada movimento do jogador.
6  ENERGIA_MOVIMENTO = 20
7
8  # Quanta energia se ganha com uma captura.
9  ENERGIA_CAPTURA = 100
10
11 # Quanto maior mais difícil a captura.
12 POSSIBILIDADE_DE_FUGA = 2
13
14 # Tamanho das casas da grelha.
15 TAMANHO_CASAS = 100
16
17
18 # Preparar a janela do jogo.
19 turtle.Screen().setup(width=700, height=700, startx=700)
20 turtle.bgcolor("orange")
21
22
23 # Desenhar a grelha.
24 t = turtle.Turtle()
25 t.hideturtle()
26 t.speed(0)
27 t.color("dark orange")
28 t.width(15)
```

```
29
30 def linha_horizontal(y):
31     t.up()
32     t.goto(-3 * TAMANHO_CASAS, y)
33     t.down()
34     t.goto(3 * TAMANHO_CASAS, y)
35
36 def linha_vertical(x):
37     t.up()
38     t.goto(x, -3 * TAMANHO_CASAS)
39     t.down()
40     t.goto(x, 3 * TAMANHO_CASAS)
41
42 # Linhas horizontais da grelha
43 linha_horizontal(3 * TAMANHO_CASAS)
44 linha_horizontal(2 * TAMANHO_CASAS)
45 linha_horizontal(1 * TAMANHO_CASAS)
46 linha_horizontal(0)
47 linha_horizontal(-1 * TAMANHO_CASAS)
48 linha_horizontal(-2 * TAMANHO_CASAS)
49 linha_horizontal(-3 * TAMANHO_CASAS)
50
51 # Linhas verticais da grelha
52 linha_vertical(3 * TAMANHO_CASAS)
53 linha_vertical(2 * TAMANHO_CASAS)
54 linha_vertical(1 * TAMANHO_CASAS)
55 linha_vertical(0)
56 linha_vertical(-1 * TAMANHO_CASAS)
57 linha_vertical(-2 * TAMANHO_CASAS)
58 linha_vertical(-3 * TAMANHO_CASAS)
59
60
61 # Criar o fugitivo com a sua imagem.
62 fugitivo = turtle.Turtle()
63 fugitivo.up()
64 turtle.register_shape("boneco-c-animal.gif")
65 fugitivo.shape("boneco-c-animal.gif")
66
67
68 # Criar o jogador com a sua imagem.
69 jogador = turtle.Turtle()
70 jogador.color("white")
71 jogador.up()
72 turtle.register_shape("boneco-c-cavaleiro.gif")
73 jogador.shape("boneco-c-cavaleiro.gif")
74
```

```
75
76 # Criar a energia do jogador.
77 energia = turtle.Turtle()
78 energia.up()
79 energia.shape("circle")
80 energia.color("gold")
81 energia.goto(0, 3 * TAMANHO_CASAS + 30)
82
83
84 # Quando executada, o fugitivo move-se ao calhas.
85 def fugitivo_foge():
86     x = random.randint(-3, 3) * TAMANHO_CASAS
87     y = random.randint(-3, 3) * TAMANHO_CASAS
88     fugitivo.goto(x, y)
89
90 def tenta_capturar(x, y):
91     jogador.goto(x, y)
92     atualiza_energia(-ENERGIA_MOVIMENTO)
93     # Captura com coordenadas iguais para o jogador e fugitivo.
94     if jogador.xcor() == fugitivo.xcor() and jogador.ycor() == fugitivo.ycor():
95         atualiza_energia(ENERGIA_CAPTURA)
96         fugitivo.circle(20)
97         fugitivo_foge()
98         return
99     # Movimentos de fuga ao calhas.
100     if random.randint(1, 6) < POSSIBILIDADE_DE_FUGA:
101         fugitivo_foge()
102
103 def atualiza_energia(quanto):
104     # Move a energia.
105     energia.forward(quanto)
106     # No limite esquerdo, derrota.
107     if energia.xcor() < -3 * TAMANHO_CASAS:
108         jogador.circle(20)
109         fim_do_jogo("Derrota...")
110     # No limite direito, ganha.
111     if energia.xcor() > 3 * TAMANHO_CASAS:
112         fim_do_jogo("Ganhaste!")
113
114 def fim_do_jogo(mensagem):
115     # Escreve mensagem junto ao jogador e as teclas deixam de fazer efeito.
116     jogador.write(mensagem, align="center", font=("Arial", 50, "bold"))
117     turtle.onkey(None, "Up")
118     turtle.onkey(None, "Down")
119     turtle.onkey(None, "Left")
120     turtle.onkey(None, "Right")
```

```
121
122 # Movimentos do jogador.
123 def para_cima():
124     x = jogador.xcor()
125     y = jogador.ycor() + TAMANHO_CASAS
126     tenta_capturar(x, y)
127
128 def para_baixo():
129     x = jogador.xcor()
130     y = jogador.ycor() - TAMANHO_CASAS
131     tenta_capturar(x, y)
132
133 def para_esquerda():
134     x = jogador.xcor() - TAMANHO_CASAS
135     y = jogador.ycor()
136     tenta_capturar(x, y)
137
138 def para_direita():
139     x = jogador.xcor() + TAMANHO_CASAS
140     y = jogador.ycor()
141     tenta_capturar(x, y)
142
143 # Prepara o programa para usar o teclado.
144 turtle.listen()
145
146 # Cada tecla executa um movimento.
147 turtle.onkey(para_cima, "Up")
148 turtle.onkey(para_baixo, "Down")
149 turtle.onkey(para_esquerda, "Left")
150 turtle.onkey(para_direita, "Right")
151
152 # Antes de iniciar, o fugitivo foge.
153 fugitivo_foge()
154
155 # Aguarda pelo teclado ou que se feche a janela.
156 turtle.mainloop()
```

Coisas que podes querer experimentar:

- Alterar as variáveis **ENERGIA_MOVIMENTO**, **ENERGIA_CAPTURA** e **POSSIBILIDADE_DE_FUGA** para ajustar a dificuldade do jogo.
- Alterar as cores do fundo, tabuleiro, e imagens do jogador e fugitivo ao teu gosto.
Para te ajudar com as imagens refere-te às pistas do capítulo anterior.
Se participaste nas nossas sessões, deverás ter, pelo menos as seguintes: “*boneco-c-vicking.gif*”, “*boneco-c-monstro.gif*”, “*boneco-a-anicas.gif*”, “*boneco-a-popas.gif*” e “*boneco-a-zombie.gif*”.

10 | Texto | Cumprimentos

Há uma tradição entre as pessoas que programam computadores: o primeiro programa que escrevem numa linguagem que estão a aprender, como por exemplo o Python, deve cumprimentar o mundo todo.

O original é em Inglês e escreve “Hello World!”.

Como nós somos Portugueses, o nosso programa vai escrever “Olá Mundo!”.

Aqui está ele:

```
1 print("Olá Mundo!")
```

Como concordarás, é um programa muito curto e simples.

Quando o executares, a mensagem “Olá Mundo!” deverá aparecer na própria janela do Mu, no painel inferior.

Uma variante personalizada do programa vai perguntar-nos o nome e depois cumprimentar-nos devidamente.

```
1 nome = input("Qual é o teu nome?")
2 print("Olá", nome)
```

Neste programa a função **input** apresenta uma pergunta: o texto que for introduzido fica depois associado à variável **nome**. Tudo isto resulta da execução da linha número 1.

Na linha seguinte, a instrução **print** começa por escrever “Olá” e seguidamente escreve o valor associado à variável **nome**: que corresponde ao nome introduzido no passo anterior.

Pista: A instrução **print** pode ser utilizada com tantos argumentos quantos quiseses, desde que sejam separados por vírgulas. Ela limita-se a escrever cada um deles no painel inferior do Mu, quando o programa está em execução. Se quiseses, podes ver alguns exemplos na tua folha **Ingredientes do Python**. Além disso, também podes utilizar a função **print** sem argumentos, só com abrir e fechar de parêntesis, por forma a deixar uma linha vazia no resultado.

11 | Texto | Pedra, Papel, ou Tesoura?

Este é um jogo que parece ter origem Chinesa, desde há muitos séculos. Será que o conheces?

Quase de certeza que sim: a **pedra** ganha à **tesoura**, que ganha ao **papel** que, por sua vez, ganha à **pedra**.

Experimenta jogar contra o computador, que usa a instrução **random.choice** para escolher a sua jogada.

Pista: O quadro chamado **Condições** na folha **Ingredientes do Python** pode ajudar-te a entender o programa.

Outra pista: Vais precisar de escrever os símbolos de abrir e fechar parêntesis rectos [e]. Algo a aprender?

```
1  import random
2
3  print("Para jogar, responde em minúsculas")
4  print("-----")
5
6  humano = input("Pedra, Papel, ou Tesoura? ")
7  computador = random.choice(["pedra", "papel", "tesoura"])
8
9  print()
10 print("A tua jogada:", humano)
11 print("A minha:", computador)
12 print()
13
14 if humano != "pedra" and humano != "papel" and humano != "tesoura":
15     print("A tua jogada é inválida.")
16
17 if humano == computador:
18     print("Empate!")
19
20 if humano == "pedra":
21     if computador == "tesoura":
22         print("Ganhaste!")
23     if computador == "papel":
24         print("Ganhei!")
25
26 if humano == "papel":
27     if computador == "pedra":
28         print("Ganhaste!")
29     if computador == "tesoura":
30         print("Ganhei!")
31
32 if humano == "tesoura":
33     if computador == "papel":
34         print("Ganhaste!")
35     if computador == "pedra":
36         print("Ganhei!")
```

12 | Números | Fazer contas

Fazer contas com o Python é bastante simples. No entanto, lembra-te que os símbolos para algumas das operações são diferentes. Podes sempre ver o quadro **Contas** na folha **Ingredientes do Python** para te ajudar.

Este programa, calcula quantos anos faltam até que a Maria, que hoje tem 12 anos, tenha 20 anos.

```
1  idade = 12
2  faltam = 20 - idade
3  print("Faltam", faltam, "anos até teres 20!")
```

O programa seria mais interessante se perguntasse a idade primeiro, e depois fizesse a conta e apresentasse o resultado. Vamos experimentar o seguinte:

```
1  idade = input("Quantos anos tens?")
2  faltam = 20 - idade
3  print("Faltam", faltam, "anos até teres 20!")
```

Se executares este programa vais ver que o Python se queixa com um erro semelhante ao seguinte:

```
Traceback (most recent call last):
  File "...", line 2, in <module>
    faltam = 20 - idade
TypeError: unsupported operand type(s) for -: 'int' and 'str'
```

Não é fácil entender estes erros! Neste caso, o Python está a queixar-se da linha 2, que corresponde à instrução **faltam = 20 - idade**: a mensagem é muito técnica, mas quer dizer que o Python não sabe fazer contas de subtrair entre números e strings (bocadinhos de texto). Porque é que isto acontece?

Porque a primeira linha associa à variável **idade** uma string (texto) com os dígitos que foram introduzidos. Para fazer a subtracção da linha seguinte, o Python precisa que esses dígitos sejam entendidos em conjunto, como um número. O programa correcto é, então:

```
1  digitos = input("Quantos anos tens?")
2  idade = int(digitos)
3  faltam = 20 - idade
4  print("Faltam", faltam, "anos até teres 20!")
```

Nota como a primeira linha associa aquilo que o utilizador escreve à variável **digitos**, que tem o texto com os dígitos correspondentes ao número. Depois, a segunda linha, constrói um número com **int(digitos)**.

Claro, o programa dará um erro se introduzirmos algo que não é um número (letras, por exemplo!), mas pelo menos funciona correctamente quando se introduzem números. É muito difícil fazer programas sem erros!

13 | Números | Ciclos de contagem

Aqui encontras um conjunto de programas que te ajudam a experimentar a instrução **for**, que permite repetir uma ou mais instruções e, de cada vez, associar a uma variável um número que começa em **0** e vai crescendo.

Este programa escreve números, contando de 0 até 9:

```
1 LIMITE = 10
2
3 for numero in range(LIMITE):
4     print("O número é", numero)
```

Com uma pequena alteração, podemos fazer um programa que conta de 1 até 10:

```
1 LIMITE = 10
2
3 for numero in range(LIMITE):
4     contagem = numero + 1
5     print("A contagem vai em", contagem)
```

Neste caso, em cada repetição do bloco de instruções agrupadas depois da linha do **for**, primeiro associamos à variável **contagem** o valor **numero + 1**, e depois utilizamos essa variável na instrução **print**. É essa conta que faz com que este programa conte de 1 até 10, em vez de contar de 0 até 9.

Com outra alteração, podemos fazer com que o programa conte de 10 até 1:

```
1 LIMITE = 10
2
3 for numero in range(LIMITE):
4     contagem = LIMITE - numero
5     print("Contagem decrescente:", contagem)
```

Experiência: Que alterações farias ao programa anterior para que ele contasse de 5 até 1? E se fosse de 5 até 0?

Este programa conta de 5 em 5 e funciona um bocado como a tabuada, já reparaste?

```
1 LIMITE = 10
2
3 for numero in range(LIMITE):
4     resultado = (numero + 1) * 5
5     print("Resultado:", resultado)
```

14 | Números e Tartaruga | Desenhar uma grelha

Uma das capacidades poderosas com a programação resulta da combinação de coisas diferentes para obter resultados mais complexos, mas com menos instruções.

Neste exemplo utilizamos os ciclos de contagem da página anterior e combinamo-los com o que sabemos do módulo **turtle** para desenhar uma grelha.

```
1  import turtle
2
3
4  ENTRE_LINHAS = 100
5  QUANTIDADE = 3
6
7  MAXIMO_VALOR = QUANTIDADE * ENTRE_LINHAS
8  QUANTAS_LINHAS = (QUANTIDADE * 2) + 1
9
10
11 t = turtle.Turtle()
12 t.hideturtle()
13 t.speed(0)
14
15
16 def linha_vertical(x):
17     t.up()
18     t.goto(x, -MAXIMO_VALOR)
19     t.down()
20     t.goto(x, MAXIMO_VALOR)
21
22 def linha_horizontal(y):
23     t.up()
24     t.goto(-MAXIMO_VALOR, y)
25     t.down()
26     t.goto(MAXIMO_VALOR, y)
27
28
29 for linha in range(QUANTAS_LINHAS):
30     coordenada = (linha * ENTRE_LINHAS) - MAXIMO_VALOR
31     linha_vertical(coordenada)
32     linha_horizontal(coordenada)
```

O programa não tem quaisquer comentários propositadamente, o que o torna difícil de entender!

Experimenta alterar a variável **ENTRE_LINHAS** para **70** e **QUANTIDADE** para **4**. Ou então, **ENTRE_LINHAS** para **30** e **QUANTIDADE** para **9**. Nota como alterando apenas esses dois valores, se desenharmos grelhas mais finas ou mais afastadas, e com mais ou menos linhas. Tudo resulta das contas incluídas no programa.

15 | Números | Adivinhar um Número

Um jogo em que o computador escolhe um número secreto entre 1 e 100 e nós temos que o adivinhar. Quando conseguirmos, o programa diz-nos quantas tentativas fizemos.

```
1  import random
2
3
4  # Computador escolhe um número secreto entre 1 e 100.
5  segredo = random.randint(1, 100)
6
7  # Este valor é de certeza diferente do segredo.
8  jogada = -1
9
10 # Para contar o número de tentativas até acertar.
11 tentativas = 0
12
13
14 # Este bloco de instruções executa repetidamente, enquanto
15 # (while em Inglês) a jogada for diferente do segredo.
16 while jogada != segredo:
17
18     # Pedir tentativa e construir um número com os dígitos.
19     digitos = input("Adivinha o número (de 1 a 100): ")
20     jogada = int(digitos)
21
22     # Aumentar o número de tentativas.
23     tentativas = tentativas + 1
24
25     # Dizer ao jogador se o número é muito pequeno / grande.
26     if jogada < segredo:
27         print("Esse é muito pequeno.")
28     if jogada > segredo:
29         print("Esse é muito grande.")
30
31
32 # Executado quando terminar o bloco de instruções acima.
33 # (quer dizer que a jogada passou a ser igual ao segredo)
34 print("Conseguiste em", tentativas, "tentativas.")
```

Este programa usa uma receita para executar grupos de instruções repetidamente que talvez não conheças: é baseada na instrução **while** (linha 16) que faz com que as instruções das linhas seguintes (as que estão 4 espaços à direita, até à linha 29) sejam executadas enquanto a condição **jogada != segredo** (linha 16) for verdadeira. Ou seja, a última instrução (linha 34) só é executada quando a **jogada** for igual ao **segredo**!

Parabéns!

Chegaste ao fim deste **Guia de Experiências**, e agora?

- Já experimentaste todos os programas?
- Já os entendeste todos?
- Já experimentaste fazer-lhes alterações?
- Já tentaste aplicar as tuas ideias a cada um?

...tudo isto são coisas que podes fazer para continuar a aprender a programar computadores com Python.

A partir de agora, com tudo o que já sabes, podes criar os teus próprios programas.

E, claro, há outras técnicas e ferramentas no Python e nos computadores que permitem fazer programas ainda mais sofisticados. Terás tempo de aprender tudo isso, se quiseres.

Até à próxima vez!