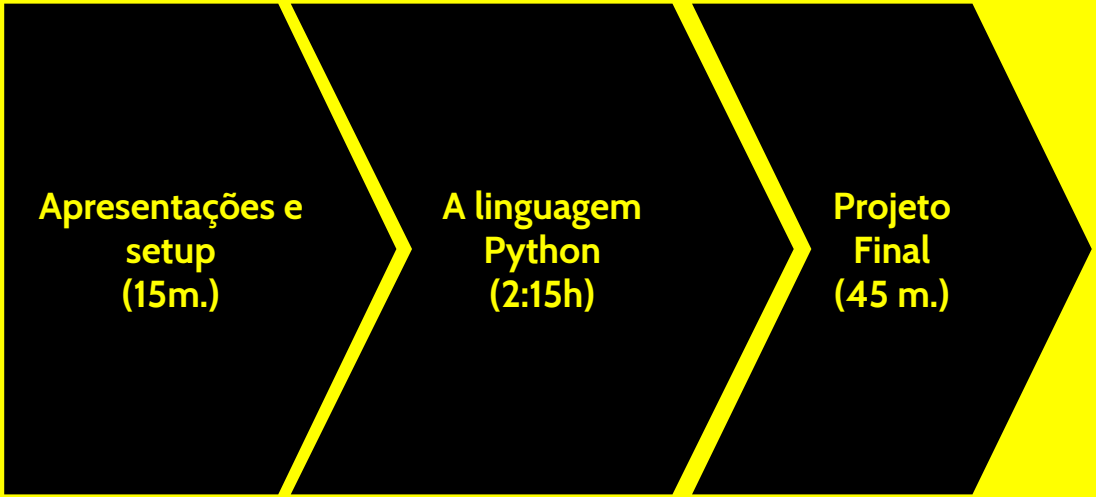
Introdução à Programação

Plano de trabalhos



Quem somos?

Quem somos?

Afonso Castro

Gustavo Silva

Setup

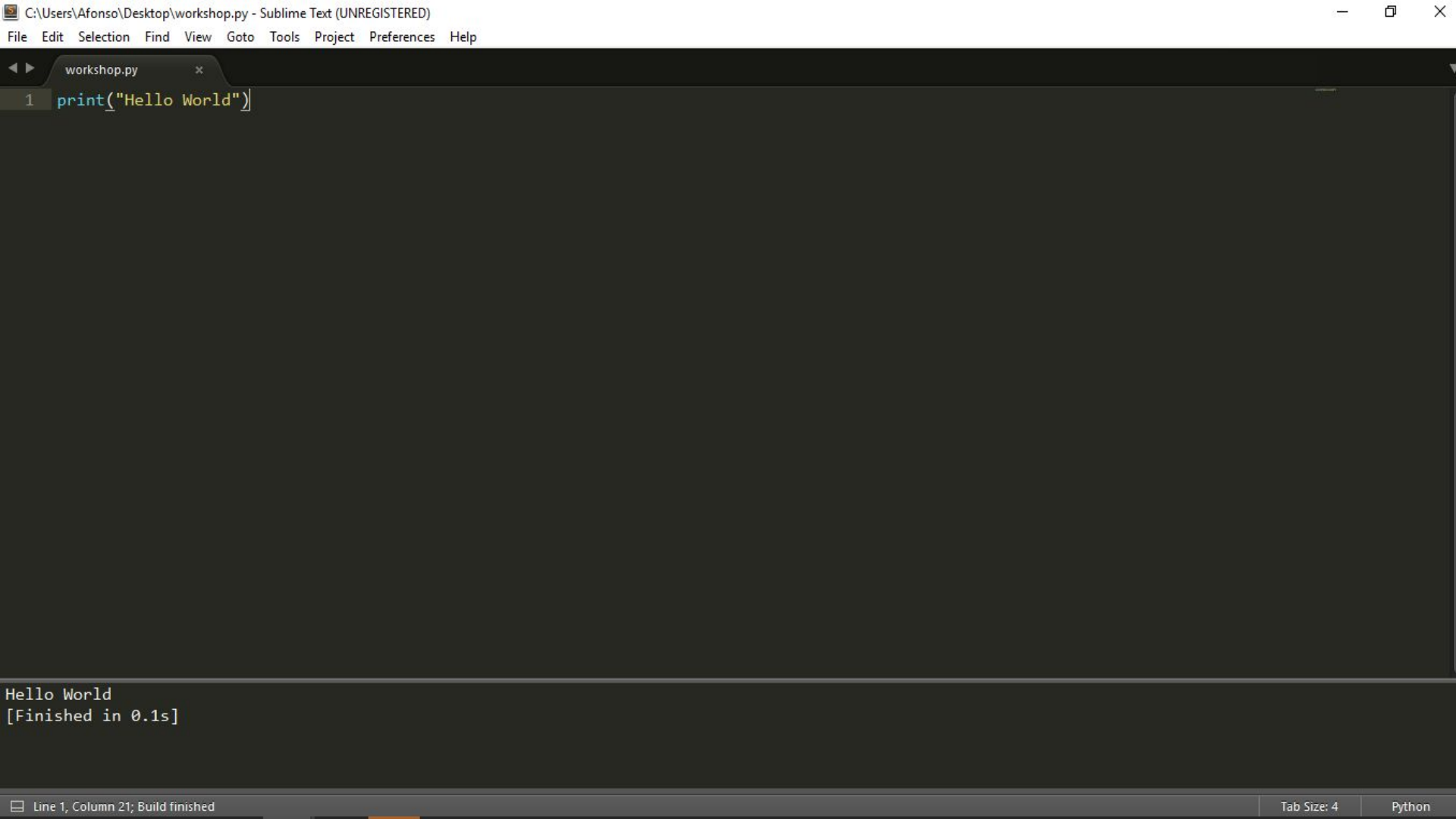
Setup

- https://github.com/sceptross/python-workshop-talkabit
- Fazer download do Python3 e PyQt (encontram-se no readme do repositório) e instalá-los.
- Um editor de texto também é necessário (link para o Sublime Text também no repositório).

Setup

- Sublime Text:
  - Tools > Build System > Python
  - Escrever print(“Hello world”)
  - No canto inferior direito, seleccionar Python
  - Guardar o ficheiro com o nome workshop.py
  - Clicar em CTRL + B para executar o código

- Outros editores de texto
  - Ativar a sintaxe de Python
  - Escrever print(“Hello world”)
  - Guardar o ficheiro com o nome workshop.py
  - Abrir a pasta onde o ficheiro foi guardado
  - Clicar em SHIFT + Botão direito do rato
  - Clicar em “Abrir janela de comando aqui”
  - Escrever python workshop.py para executar o código


C:\Users\Afonso\Desktop\workshop.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
workshop.py
1 print("Hello World")
Hello World
[Finished in 0.1s]
Line 1, Column 21; Build finished
Tab Size: 4
Python

Setup

C:\Users\Afonso\Desktop\workshop.py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

workshop.py

```
1  print("Hello World")
```

C:\Windows\system32\cmd.exe

```
C:\Users\Afonso\Desktop>python workshop.py
Hello World

C:\Users\Afonso\Desktop>
```

Line 1, Column 21 Tab Size: 4 Python

workshop.py

A Linguagem Python

Capítulos

1ª Parte (1:20h)

1. Sintaxe (15 minutos)
2. Strings e Prints (10 minutos)
3. Controlo de Fluxo (15 minutos)
4. Funções (20 minutos)
5. Listas e dicionários (20 minutos)

2ª parte: (0:55h)

6. Ciclos (25 minutos)
7. Classes (30 minutos)

1

Sintaxe

Sintaxe

- Variáveis
  - Guardam valores para utilização futura.
- Comentários
  - Tornam o código mais simples de ler.
    - Um # denota o início de um comentário
- Operadores aritméticos
  - Permitem realizar cálculos matemáticos.

Tipos de Variáveis

- Inteiro
  - Guarda números inteiros.
    - x = 5
- Decimal
  - Guarda números decimais.
    - x = 5.5
- Booleano
  - Guarda valores do tipo *True* ou *False*.
    - x = True
- String
  - Guarda texto.
    - x = “Talk a Bit”
- Existem mais tipos, mas falaremos deles mais tarde!

Tipos de operações aritméticas

- Soma (+)
  - x = 6 + 5 # x = 11
- Subtração (-)
  - x = 6 - 5 # x = 1
- Multiplicação (*)
  - x = 6 * 5 # x = 30
- Divisão (/)
  - x = 6 / 5 # x = 1.2
- Divisão inteira (//)
  - x = 6 // 5 # x = 1
- Resto (%)
  - x = 6 % 5 # x = 1
- Potência (**)
  - x = 6 ** 5 # x = 7776
- Seguem-se as regras de prioridades da matemática, podendo recorrer-se também aos parênteses para as alterar.

Exercício 1

- Escrever um comentário na primeira linha. Pode conter qualquer texto.
- Nas linhas seguintes:
  - Criar uma variável chamada inteiro e dar-lhe o valor 5.
  - Criar uma variável chamada decimal e dar-lhe o valor do resultado da divisão de inteiro por 2.
  - Criar uma variável chamada potencia e dar-lhe o valor de inteiro elevado a 3.
  - Criar uma variável chamada string com o texto “A minha primeira string”.
  - Criar uma variável chamada booleano que deverá conter o valor de verdade da pergunta “É a primeira vez que estou a programar?”.

Exercício 1 (solução)

```
# Make Talk a Bit Great Again!
inteiro = 5
decimal = inteiro / 2
potencia = inteiro ** 3
string = "A minha primeira string"
booleano = False
```

2

Strings e prints

Operações sobre strings

- len()
  - Retorna o número de carateres de uma string
    - len(“Python”) = 6
- lower()
  - Converte todas as letras maiúsculas da string em letras minúsculas
    - “Python”.lower() # “python”
- upper()
  - Converte todas as letras minúsculas da string em letras maiúsculas
    - “Python”.upper() # “PYTHON”
- str()
  - Converte não-strings em strings.
    - str(15) # “15”

Notação

- Porque é que algumas operações usam uma sintaxe com ponto e outras com parênteses?
  - Operações com ponto apenas podem ser usadas nesse tipo.
  - Operações com parênteses podem ser usadas com qualquer tipo.

Prints

- E se quisermos mostrar informação ao utilizador do nosso programa?
  - print(“Python”)
  - print(x)
  - print(“2+2=”, str(2+2))
  - print(“A minha linguagem de programação favorita é”, “Python”)

Exercício 2

- Criar uma variável chamada a_minha_segunda_string e dar-lhe o valor de “segunda”.
- Imprimir (print) o tamanho da string “Python”
- Imprimir o valor da variável a_minha_segunda_string com todas as letras maiúsculas.
- Imprimir a frase seguinte, recorrendo ao operador de resto: “Dividir 6 por 4 tem um resto de 2”.

Exercício 2 (solução)

```
a_minha_segunda_string = "segunda"
print(len(a_minha_segunda_string))
print(a_minha_segunda_string.upper())
print("Dividir 6 por 4 tem um resto de 2", str(6 % 4))
```

3

Controlo de fluxo

Comparadores

- **==**
  - “Igual a” (=)
- **!=**
  - “Diferente de” (≠)
- **<**
  - “Menor que”
- **>**
  - “Maior que”
- **<=**
  - “Menor ou igual que”
- **>=**
  - “Maior ou igual que”

Operadores booleanos

P	Q	P and Q
V	V	V
V	F	F
F	V	F
F	F	F

Operadores booleanos

P	Q	P or Q
V	V	V
V	F	V
F	V	V
F	F	F

Operadores booleanos

P	not P
V	F
F	V

Operadores booleanos - exemplos

- (4 >= 5) and (5 < 7)
  - False
- (4 >= 5) or (5 < 7)
  - True
- not (4 == 4)
  - False
- x = 5
- not x > 6 and x < 4
  - True
- 4 > 5 or (5 > 4 and 6 > 5)
  - True
- 4 > 5 or 5 > 4 and 6 > 5
  - ???

Operadores booleanos

- Tal como nos operadores matemáticos, também existe uma regra de prioridade para os operadores booleanos.
- **not** tem prioridade sobre **and**, que tem prioridade sobre **or**.
- Tal como nos operadores matemáticos, também é possível usar parênteses para alterar as prioridades.

Controlo de fluxo

- **if** *expressão*:
  - Avalia se *expressão* tem valor de verdade True e em caso afirmativo executa o código correspondente.
- **else**
  - Se *expressão* tem valor de verdade False, o código correspondente ao **else** é executado.
- **elif**
  - Equivalente a um **else** seguido de um **if.**

```
1  if 4 > 5:
2      print("Wait what?")
3      print("You don't know math, pal")
4  elif 4 == 5:
5      print("No comments.")
6  else:
7      print("Great, you know that 5 is greater than 4. Want a biscuit?")
8  print("End of the program")
```

Scope

- Como é que se determina que código é que corresponde a um **if**, a um **elif** ou a um **else** (ou seja, o seu *scope*)? Tabulações.
- Existem mais situações em que é necessário delimitar o *scope* de uma operação, mas serão abordados mais à frente.
- É possível encadear um **if** dentro de outro **if** (e o mesmo se aplica a todas as operações que necessitem que se delimite o seu *scope*).

```
1  if 4 > 5:
2      print("Wait what?")
3      print("You don't know math, pal")
4  elif 4 == 5:
5      print("No comments.")
6  else:
7      print("Great, you know that 5 is greater than 4. Want a biscuit?")
8  print("End of the program")
```

Exercício 3

Converter o seguinte excerto do poema “If..” de Rudyard Kipling para código, fazendo uso de variáveis, comparações e controlo de fluxo, de forma a que, se todas as condições no poema se aplicarem, os dois últimos versos sejam impressos no ecrã. Criar uma variável para cada uma das condições (ex: you_can_talk_with_crowds = True).

If *you can talk with crowds* ***and*** *keep your virtue,*
Or *walk with kings - **nor*** [or not] *lose the common touch,*
If ***neither*** *foes* ***nor*** *loving friends can hurt you,*
If *all men count with you, but* ***none*** *too much* [max_confidence = 100];

(...)
Yours is the Earth and everything that's in it,
And - which is more - you'll be a Man my son!

Exercício 3 (solução)

```
you_can_talk_with_crowds = True
you_can_keep_your_virtue = True
walk_with_kings = True
lose_the_common_touch = True
foes_can_hurt_you = False
loving_friends_can_hurt_you = False
all_men_count_with_you = True
max_confidence = 99

if (you_can_talk_with_crowds and you_can_keep_your_virtue) or
walk_with_kings or not lose_the_common_touch:
     if not foes_can_hurt_you and not loving_friends_can_hurt_you:
          if all_men_count_with_you and max_confidence < 100:
               print("Yours is the Earth and everything that's in it," )
               print("And - which is more - you'll be a Man my son!" )
```

4

Funções

Funções

- Permitem reutilizar código, sem termos de o reescrever outra vez, sempre que precisarmos dele.
- Tal como uma função matemática, recebem argumentos (objetos) e retornam algo (imagem)
- Uma função é composta por um cabeçalho, onde se define o nome da função e os argumentos da função. O cabeçalho começa sempre pela keyword **def**.
  - def potencia(base, expoente):
- Para além do cabeçalho, também é composta por um corpo, onde se descreve o conteúdo da função, bem como o seu retorno (recorrendo à *keyword* **return)**.
  - return base ** expoente

```
1  def potencia(base, expoente):
2      resultado = base ** expoente
3      return resultado
```

Utilizar Funções

- As funções não são executadas se não forem chamadas.
- Para chamar uma função, basta usar a nomenclatura nome(arg1, arg2,...)
  - potencia(2, 3)
- O retorno das funções pode ser guardado em variáveis.

```
1  def potencia(base, expoente):
2      resultado = base ** expoente
3      return resultado
4
5  resultado = potencia(2, 3)
6  print resultado
```

```
8
[Finished in 0.4s]
```

Módulos

- E se quisermos calcular a raíz quadrada de um número?
  - ✎ Experimentar calcular sqrt(25)
- O Python não sabe o que é uma raíz quadrada. No entanto, existe um módulo chamado **math** que acrescenta ao Python várias funções e constantes matemáticas.
- Tal como o módulo math, existem outros módulos, e é inclusive possível criamos os nossos próprios módulos.

Importar Módulos

- Para importar módulos recorremos às *keywords* **import** e **from.**
  - import math
    - Importa o módulo **math**. Para usarmos a função sqrt do módulo, temos de usar math.sqrt().
  - from math import sqrt
    - Importa a função sqrt do módulo **math**. Basta usar sqrt() para usar a função. Podemos importar mais funções, recorrendo à vírgula
      - From math import sqrt, cos
  - from math import *
    - Importa o módulo todo, mas não é preciso referir o nome do módulo na chamada da função.

Funções No Python

- Existem algumas funções que já estão incorporadas no Python, sem ser preciso recorrer a módulos.
  - Algumas já vimos, como len(), que nos dava o tamanho de um objeto e str() que convertia um objeto para string.
- Entre estas funções, destacam-se o max(), min(), que recebem um número variável de argumentos e retornam o maior e o menor, respetivamente, e o type(), que recebe um objeto e retorna o seu tipo.
  - print(type(42) == int) # True
  - print(type(42)) # <type 'int'>
  - print(max(1,2,3,4)) # 4

Exercício 4

- Criar uma função de nome *calcular* que recebe três argumentos - um de nome *operacao*, outro de nome *operador1* e outro de nome *operador2*.
  - Se operacao for a string “+”, “-”, “*”, “/”, “%” ou “**”, “max”, “min” fazer a soma, subtração, multiplicação, divisão, resto, potência máximo ou mínimo de *operador1* e *operador2*, respetivamente.
  - Senão, imprimir a mensagem “Erro!” e retornar o tipo do argumento *operacao*.
- Criar uma função de nome *distancia_entre_pontos* que recebe quatro argumentos (x1, y1, x2, y2) e calcula a distância entre os pontos 1 e 2.

Exercício 4 (solução)

```
import math

def calcular(operacao, operador1, operador2):
        if operacao == "+":
                return operador1 + operador2
        elif operacao == "-":
                return operador1 - operador2
        elif operacao == "*":
                return operador1 * operador2
        elif operacao == "/":
                return operador1 / operador2
        elif operacao == "%":
                return operador1 % operador2
        elif operacao == "**":
                return operador1 ** operador2
        elif operacao == "max":
                return max(operador1, operador2)
        elif operacao == "min":
                return min(operador1, operador2)
        else:
                print("Erro!")
                return type(operacao)

def distancia_entre_pontos(x1, y1, x2, y2):
        return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
```

5

Listas e dicionários

Listas

- Listas são sequências de informação, guardadas numa só variável.
  - animais = [“gato”, “cao”, “peixe”, “passaro”]
- Todos os elementos de uma lista têm um índice numérico associado. Podemos aceder a um elemento de uma lista pelo seu índice. Notar que os índices começam por 0.
  - print animais[0] # gato
  - print animais[1] # cao
- Para substituir um elemento de uma função por outro, basta aceder-lhe pelo seu índice e alterar o seu valor como se fosse uma variável.
  - animais[2] = “lobo”

Funções e Capacidades de uma Lista

- E se quisermos adicionar um item a uma lista, sem necessariamente substituir por outro?
  - list.append(item)
- len() funciona para listas, tal como para strings
- list.sort() permite-nos ordenar uma lista.
  - Números são ordenados por ordem crescente
  - Strings são ordenadas por ordem alfabética
- list.index(item) retorna o índice do primeiro elemento na lista igual a *item*.
- list.insert(n, item) insere o elemento *item* na posição *n*, colocando os elementos seguintes no índice i+1, sendo i o seu índice antes da inserção.

List Slicing

- Às vezes, apenas queremos aceder a parte de uma lista.
- Para o fazermos, recorremos à mecânica *list slicing* do Python:
  - animais = [“gato”, “cao”, “peixe”, “passaro”, “lobo”]
    - print animais[1:3] # [“cao”, “peixe”]
    - print animais[3:] # [“passaro, “lobo”]
    - print animais[:3] # [“gato”, “cao”]
    - print animais[:-2] # [“gato”, “cao”, “peixe”]
    - print animais[:] # [“gato”, “cao”, “peixe”, “passaro”, “lobo”]
- É possível usar *list slicing* numa string
  - Aliás, uma string pode ser vista e usada como uma lista de carateres.

Exercício 5

- Criar uma lista com 5 desportos.
  - Imprimir os 3 primeiros elementos dessa lista, usando apenas uma linha de código.
  - Substituir o desporto preterido da lista por um desporto diferente.
- Criar uma função que receba uma lista, a ordene e retorne o primeiro elemento.
- Criar uma função que receba uma lista e imprima a frase “O meu primeiro elemento é um número”, “O meu primeiro elemento é uma string” ou “O meu primeiro elemento não é um número nem uma string”, consoante o seu primeiro elemento for um número, uma string ou de outro tipo.

Exercício 5 (solução)

```
desportos = ["Futebol", "Basquetebol", "Andebol", "Voleibol", "Judo"]
print(desportos[:3])
desportos[3] = "Voleibol de praia"

def f1(lista):
        lista.sort()
        return lista[0]

def f2(lista):
        if type(lista[0]) == int:
                print("O meu primeiro elemento e um numero")
        elif type(lista[0]) == str:
                print("O meu primeiro elemento e uma string")
        else:
                print("O meu primeiro elemento nao e um numero nem uma string")
```

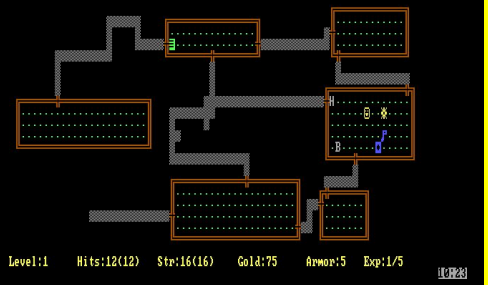
Dicionários

- Dicionários são listas que em vez de terem um índice numérico, têm um índice, chamado **chave**, à nossa escolha.
  - numeros = {“um”: 1, “dois”: 2, “tres”: 3}
- Os elementos associados a cada **chave** têm o nome de **valor**.
- Um dicionário não pode ter chaves duplicadas.

Manipular Dicionários

- Para se adicionar um elemento a um dicionário, basta fazer dicionario[chave] = valor
  - numeros[“quatro”] = 4
- Elementos podem ser removidos de um dicionário com
  - **del** dicionario[chave]
- Para se substituir um elemento, é como inserir um novo elemento (sendo a anterior entrada no dicionário apagada e a nova inserida).

Exercício 6



- Rogue é um jogo de computador, criado em 1980, que deu origem a um género de jogos chamado *roguelikes*.
- O jogo passava-se numa gruta com 24 andares, em que o objetivo era chegar ao fundo, obter o amuleto de Yendor e depois voltar ao topo sem morrer.
- Ao longo da gruta, podiam apanhar-se vários itens, a maior parte deles deixados por aventureiros em busca do lendário amuleto que não tiveram muita sorte.

Exercício 6 (cont.)

- Vamos tentar replicar o sistema de inventário do Rogue:
  - Criar um dicionário de nome inventario.
  - Criar uma chave de nome “gold”, que representa o ouro atual do jogador, com o valor 400.
  - Criar uma chave chamada “weapons” com as seguintes armas: “Mace”, “Long Sword”, “Spear”.
  - Criar uma chave chamada “armors” com as seguintes armaduras: “Leather”, “Ring mail”, “Chain Mail”.
  - Criar uma chave chamada “potions” com as seguintes poções: “Poison”, “Healing”, “See Invisible”.
  - Criar uma chave chamada “tem_amuleto” com o valor False.

Exercício 6 (cont.)

- Vamos agora criar algumas funções que nos permitem modificar o inventário:
  - Criar uma função encontrou_amuleto() que recebe um inventário e coloca o amuleto no inventário.
  - Criar uma função use_potion() que recebe um inventário e um nome de poção e consome essa poção.
  - Criar a função que adiciona uma nova arma ao inventário.
  - O jogador encontrou um scroll de nome “Scare Monster”. Criar a função que adiciona uma nova key ao dicionário para os scrolls e lhe adiciona o novo scroll. É possível ter-se mais de um scroll.
  - O jogador encontrou ouro. Criar uma função adicionar_ouro() que recebe um inventário e uma quantidade de ouro e o adiciona ao inventário.

Exercício 6 (solução)

```
inventario = {}
inventario["gold"] = 400
inventario["weapons"] = ["Mace", "Long Sword", "Spear"]
inventario["armors"] = ["Leather", "Ring mail", "Chain Mail"]
inventario["potions"] = ["Poison", "Healing", "See Invisible"]
inventario["tem_amuleto"] = False

def encontrou_amuleto(inventario):
    inventario["tem_amuleto"] = True

def use_potion(inventario, nome):
    inventario["potions"].remove(nome)

def add_weapon(inventario, nome):
    inventario["weapons"].append(nome)

def add_scroll(inventario, nome):
    inventario["scrolls"] = ["Scare Monster"]

def adicionar_ouro(inventario, quantidade):
    inventario["gold"] += quantidade
```

6

Ciclos

Ciclos

- Um ciclo permite-nos executar um pedaço de código várias vezes seguidas.
- Um ciclo **while** é semelhante a um **if**, só que em vez de executar um pedaço de código **se** uma condição for verdade, executa um pedaço de código **enquanto** uma condição for verdade.

```
1  i = 0
2  while i < 10:
3      print("Ainda estamos no ciclo")
4      i += 1
5  print("Saimos do ciclo")
```

- Neste pedaço de código são realizadas várias operações:
  - i é inicializado a 0 e posteriormente avaliado.
  - Como é inferior a 10, o código continua a ser executado.
  - No código que se encontra no *scope* do ciclo **while**, podemos ver a operação i += 1. Esta é a operação que nos vai permitir sair do ciclo.

Ciclo While

```
1  i = 0
2  while i < 10:
3      print("Ainda estamos no ciclo")
4  print("Saimos do ciclo")
```

- Este ciclo não tem uma forma de atingir a sua **condição de paragem**. Estamos perante um ciclo infinito.
- Há duas formas de isto acontecer:
  - A condição do ciclo é sempre verdadeira.
  - O corpo do ciclo não permite à condição ser falsa
- E se quisermos sair do ciclo antes da condição de paragem ser atingida?
  - A *keyword* **break** permite-nos sair de um ciclo a qualquer momento.
  - Principalmente útil quando temos mais que uma condição de paragem ou ela é demasiado complexa.

Ciclo While / Else

- Um ciclo **while...else** é exatamente igual a um ciclo **while**, com a diferença que depois de se sair do ciclo while, se a condição do ciclo for falsa, o pedaço de código referente ao **else** é executado.

```
1  import random
2
3  print "Lucky Numbers! 3 numbers will be generated."
4  print "If one of them is a '5', you lose!"
5
6  count = 0
7  while count < 3:
8      num = random.randint(1, 6)
9      print num
10     if num == 5:
11         print "Sorry, you lose!"
12         break
13     count += 1
14 else:
15     print "You win!"
```

Ciclo For

- Um ciclo **for** é um outro tipo de ciclo, geralmente utilizado quando sabemos o número de vezes que queremos executar o código dentro do ciclo. Este ciclo requer que se percorra um objeto iterável (ou seja, um conjunto de objetos, como listas, strings ou dicionários).
  - for i in iteravel:
    - Lê-se “para cada i em iteravel”

```
1  numbers = [1, 2, 3, 4, 5, 6]
2
3  for number in numbers:
4      print(number ** 2)
5
6
```

```
1
4
9
16
25
36
[Finished in 0.1s]
```

Ciclo For

- Tal como para o ciclo **while**, existe o ciclo **for...else**, onde o código no *scope* do **else** é executado se o ciclo for terminado da forma esperada (ou seja, sem recurso ao **break**).
- Iterar por dicionários resulta em obtermos, a cada iteração do ciclo, uma das chaves do dicionário.

```
1  numbers = {"one": 1, "two": 2, "three": 3}
2
3  for key in numbers:
4      print("A chave e: " + str(key))
5      print("O valor e: " + str(numbers[key]) + '\n')
6
```

```
A chave e: three
O valor e: 3

A chave e: two
O valor e: 2

A chave e: one
O valor e: 1

[Finished in 0.1s]
```

Ciclo For

- Uma das desvantagens do ciclo **for** é não termos acesso ao índice do objeto para que estamos a olhar. Para remediar isto, existe a função enumerate() do Python, que recebe um iterável e retorna o índice do objeto e o próprio objeto. Também é possível criar uma lista com todos os números de 0 a n, recorrendo a range(n)

```
1  for indice, item in enumerate(lista):
2      print indice, item
```

- Se quisermos iterar por mais que uma lista, podemos juntá-las recorrendo ao zip().

```
1  lista_a = [3, 9, 17, 15, 19]
2  lista_b = [2, 4, 8, 10, 30, 40, 50, 60, 70, 80, 90]
3
4  for a, b in zip(lista_a, lista_b):
5      print(max(a,b), end=' ')
6
7
```

```
3 9 17 15 30
[Finished in 0.1s]
```

Exercício 7

- Existe um monstro no Rogue (Leprechaun), que quando executa com sucesso um ataque rouba ouro ao jogador. Criar uma função steal_gold(), que recebe o inventário do jogador, escolhe um número aleatório x entre 1 e 4 e de 1 em 1 segundo, rouba uma quantidade aleatória de gold entre 50 e 200. Algumas considerações para este exercício:
  - Para se gerar um número aleatório, é preciso importar o módulo random e utilizar a função randint(min, max), que gera um número aleatório entre min e max, **inclusivé**.
  - Para se esperar 1 segundo, é preciso importar o módulo time e utilizar a função sleep(n), em que n representa o número de segundos que se quer esperar.
  - Sempre que o ouro do jogador é diminuído, a mensagem “Oh no! You lost X gold!” é mostrada no ecrã.
  - Se nalgum momento o ouro do jogador chegar a 0, a mensagem “You lost all your gold!” deve ser mostrada e o ciclo parado. Se no fim do ciclo isso não tiver acontecido, a mensagem “At least you still have X gold...” deve ser mostrada.

Exercício 7 (cont.)

- Criar uma função show_inventory() que percorre o inventario e imprime no ecrã as listas de itens (as armas, poções, armadura e scrolls do jogador). O formato deve ser o seguinte:
  - Índice) [Tipo do item] called [nome do objeto]
    - Ex: 1) Weapon called Long Sword
  - Notas:
    - É possível juntar duas listas recorrendo ao operador de soma.
    - É obrigatório o uso da função enumerate().
    - É obrigatório percorrer o dicionário.
    - A função capitalize() de uma String retorna a String com o primeiro carater maiúsculo.

Exercício 7 (solução)

```
import random
import time

def steal_gold(inventory):
        x = random.randint(1, 4)
        while x > 0:
                stolen_gold = random.randint(50, 200)
                print("Oh no! You lost", stolen_gold, "gold!")
                inventory["gold"] -= stolen_gold
                time.sleep(1)
                x -= 1
                if inventory["gold"] <= 0:
                        inventory["gold"] = 0
                        print("You lost all your gold!")
                        break
        else:
                print("At least you still have", inventory["gold"] ,"gold...")

def show_inventory(inventory):
        all_items = []
        for key in inventory:
                if type(inventory[key]) == list:
                        all_items += inventory[key]
                        for index, item in enumerate(all_items):
                                print(str(index+1) + ") " + key.capitalize()[:-1] + " called", item)
```


7
Classes

Classes

- Classes permitem-nos modelar sob a forma de objetos aquilo que pretendemos programar.
- Uma classe representa uma caracterização de um objeto sob a forma de um novo tipo (por exemplo, classe Pessoa). Depois de criarmos a classe, seremos capazes de criar variáveis deste novo tipo.
- Cada classe tem associadas variáveis (que representam características da classe) e funções (que representam acções sobre a classe).
- Quando terminarmos a classe e começarmos a criar variáveis do novo tipo, cada uma dessas variáveis terá uma cópia sua de cada uma das variáveis e funções.

Classes

- Todas as classes devem ter um construtor, que será o código a ser executado quando o programador cria uma nova variável do novo tipo.
- Ao programarmos a classe, temos frequentemente de nos referir à hipotética variável do novo tipo. O primeiro argumento em cada função e no construtor representa o próprio objeto.

```python
class Person():
    def __init__(self, first_name, last_name, age):
        self.first_name = first_name
        self.last_name = last_name
        self.age = age

    def print_full_name(self):
        print("Hello! My name's " + self.first_name + " " + self.last_name)

    def print_age(self):
        print("I'm " + str(self.age) + " years old.")

    def is_child(self):
        if self.age < 18:
            print("I'm a child!")
            return True
        else:
            print("I'm a grown person.")
            return False

    def change_last_name(self, new_last_name):
        self.last_name = new_last_name
```

```
23
24
25  afonso = Person("Afonso", "Castro", 20)
26  afonso.print_full_name()
27  afonso.print_age()
28  afonso.change_last_name("Silva")
29  afonso.print_full_name()
```

```
Hello! My name's Afonso Castro
I'm 20 years old.
Hello! My name's Afonso Silva
[Finished in 0.1s]
```

Exercício 8

- Queremos guardar a informação referente a um campeonato de Fórmula 1 numa classe Campeonato:
  - Num campeonato de Fórmula 1, há pilotos. Sobre os pilotos, queremos saber o primeiro e último nome.
  - Para além dos pilotos, há equipas. Cada equipa tem um nome e dois pilotos (piloto1 e piloto2).
  - Também existem corridas. Sobre as corridas, queremos saber o nome, o comprimento e o país de localização. Também queremos manter uma lista ordenada de pilotos (para os resultados).
  - Na classe Campeonato, interessa guardar as equipas, as corridas e a classificação dos pilotos. A cada piloto corresponde uma pontuação.
- Criar as classes necessárias para modelar os possíveis objetos deste enunciado, bem como construtores e as suas variáveis.

Exercício 8 (cont.)

- Vamos criar algumas funções para as classes:
  - Criar uma função que permita imprimir o nome de um piloto.
  - Criar uma função que obtém de todas as corridas a sua localização, as coloca numa lista e ordena essa lista por ordem alfabética.
  - Criar uma função que adiciona à classificação de pilotos um novo piloto, colocando a 0 o valor da sua pontuação.
  - Criar uma função que coloque numa lista os pilotos com mais de 50 pontos e a retorne.
  - ...

Exercício 8 (solução)

```
class Campeonato():
    def __init__(self):
        self.classificacao = {};
        self.corridas = [];
        self.equipas = [];

    def localizacoes(self):
        result = []
        for corrida in self.corridas:
            result.append(corrida.pais)
        result.sort()
        return result;

    def melhores_pilotos(self):
        result = []
        for piloto in self.pilotos:
            if (classificacao[piloto] > 50):
                result.append(classificacao[piloto])
        return result

    def adicionar_piloto(self, piloto):
        self.pilotos[piloto] = 0
```

Exercício 8 (solução)

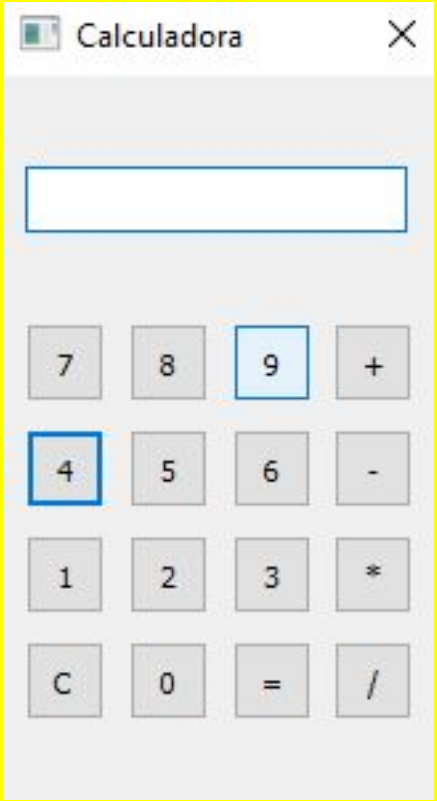
```
class Piloto():
    def __init__(self, primeiro_nome, ultimo_nome):
        self.primeiro_nome = primeiro_nome
        self.ultimo_nome = ultimo_nome
        self.pontuacao = 0

    def print_full_name(self):
        print(self.primeiro_nome, self.ultimo_nome)

class Equipa():
    def __init__(self, nome, piloto1, piloto2):
        self.nome = nome
        self.pilotos = [piloto1, piloto2]

class Corrida():
    def __init__(self, nome, comprimento, pais):
        self.nome = nome
        self.comprimento = comprimento
        self.pais = pais
        self.pilotos = []
```

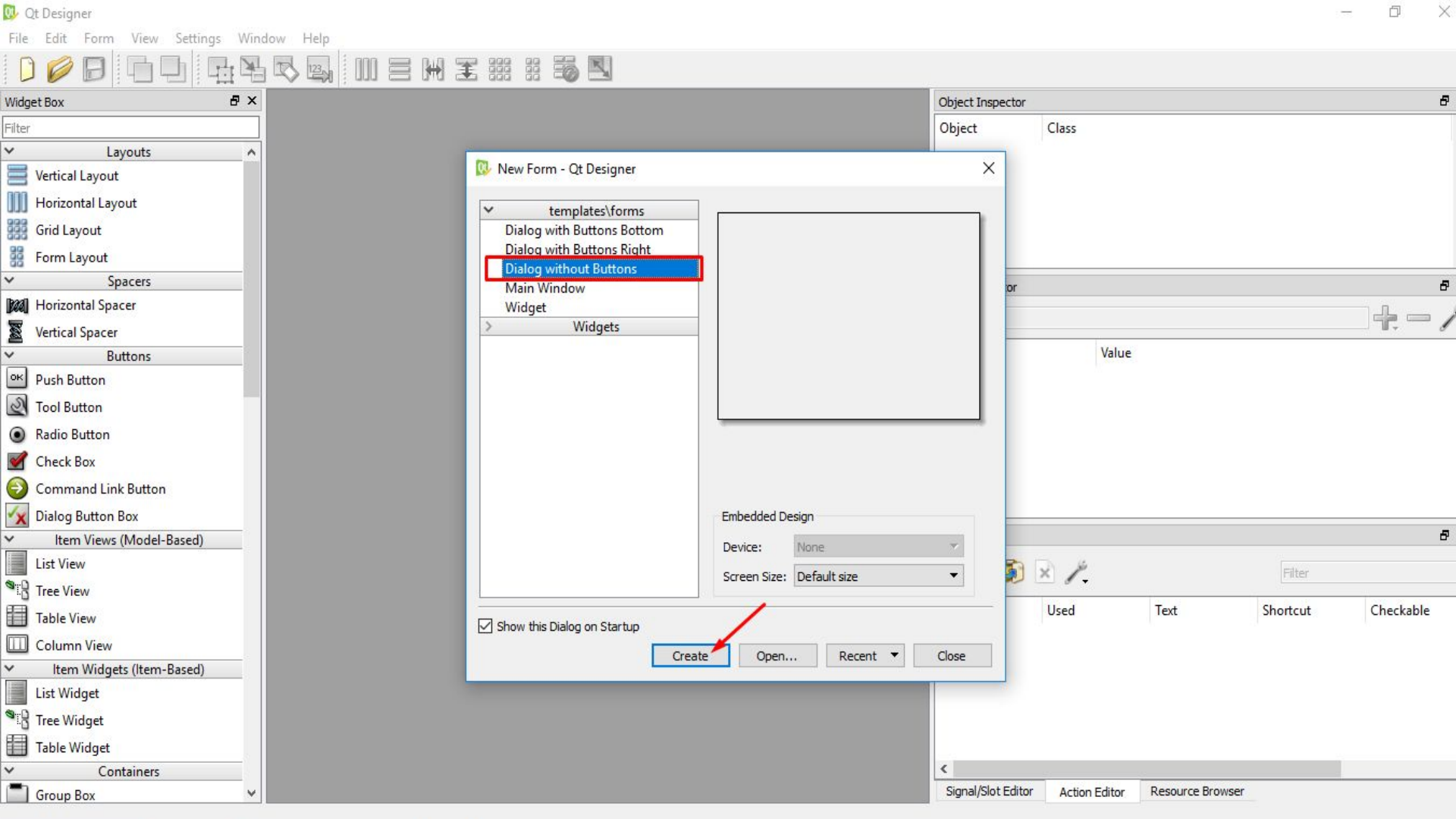
Projeto Final

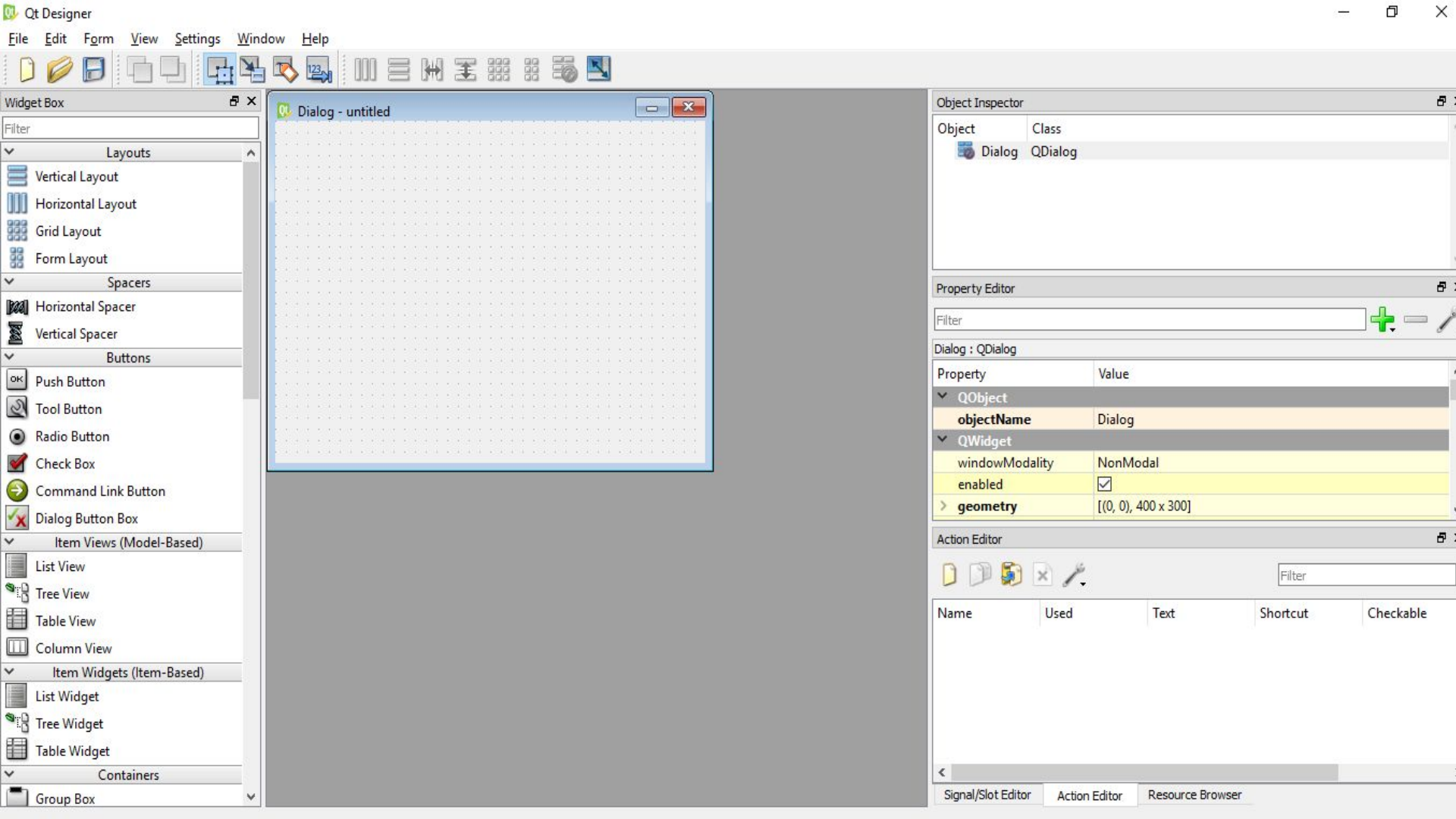
Especificação

- Criar uma calculadora “de merceeiro”, recorrendo a uma GUI (Graphical User Interface).
- A calculadora deverá ter as operações de soma, subtração, multiplicação e divisão.
- Deverá ter botões para:
  - Introduzir os 10 dígitos
  - Selecionar a operação pretendida (+,-,x,/)
  - Calcular a operação (=)
  - Limpar o ecrã (bem como “esquecer” cálculos em curso) (C)
- Deverá também ter um visor para mostrar os números enquanto eles são introduzidos, bem como o resultado.

Aspeto Final

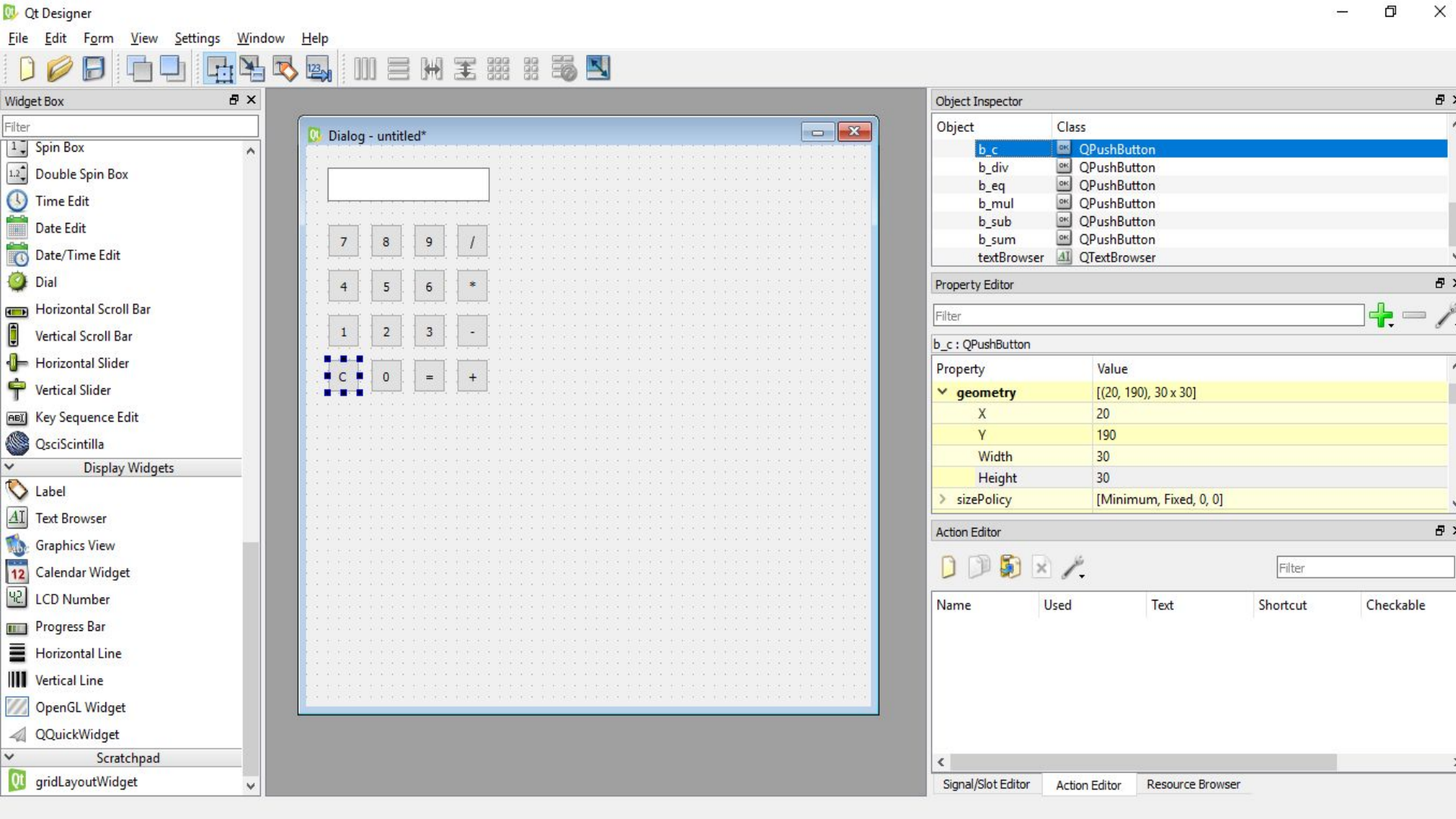



Qt Designer
File Edit Form View Settings Window Help
Widget Box
Filter
Layouts
Vertical Layout
Horizontal Layout
Grid Layout
Form Layout
Spacers
Horizontal Spacer
Vertical Spacer
Buttons
Push Button
Tool Button
Radio Button
Check Box
Command Link Button
Dialog Button Box
Item Views (Model-Based)
List View
Tree View
Table View
Column View
Item Widgets (Item-Based)
List Widget
Tree Widget
Table Widget
Containers
Group Box
Object Inspector
Object
Class
Value
Used
Text
Shortcut
Checkable
Signal/Slot Editor
Action Editor
Resource Browser
New Form - Qt Designer
templates\forms
Dialog with Buttons Bottom
Dialog with Buttons Right
Dialog without Buttons
Main Window
Widget
Widgets
Embedded Design
Device:
None
Screen Size:
Default size
Show this Dialog on Startup
Create
Open...
Recent
Close


Qt Designer
File Edit Form View Settings Window Help
Widget Box
Filter
Layouts
Vertical Layout
Horizontal Layout
Grid Layout
Form Layout
Spacers
Horizontal Spacer
Vertical Spacer
Buttons
Push Button
Tool Button
Radio Button
Check Box
Command Link Button
Dialog Button Box
Item Views (Model-Based)
List View
Tree View
Table View
Column View
Item Widgets (Item-Based)
List Widget
Tree Widget
Table Widget
Containers
Group Box
Dialog - untitled
Object Inspector
Object
Class
Dialog
QDialog
Property Editor
Filter
Dialog : QDialog
Property
Value
QObject
objectName
Dialog
QWidget
windowModality
NonModal
enabled
geometry
[(0, 0), 400 x 300]
Action Editor
Filter
Name
Used
Text
Shortcut
Checkable
Signal/Slot Editor
Action Editor
Resource Browser

Desenhar a GUI

- Criar 16 botões do tipo "*Push Button*" (secção Buttons).
- Criar um "*TextBrowser*" (secção Display Widgets)
- Selecionar os 16 botões e na janela "*Property Editor*", na secção "*Geometry*", mudar a "*height*" e a "*width*" para 30. Seleccionar o "*TextBrowser*" e mudar a "width" para 30.
- Aproximar os botões de forma a dispô-los em matriz e ajustar o comprimento do "*TextBrowser*" para se ajustar ao comprimento total dos botões.
- Alterar o texto dos botões de forma a que corresponda a uma calculadora convencional.
- Na janela "*Object Inspector*", alterar o nome dos botões para b_X, sendo X o texto/operação do botão.


Qt Designer
File Edit Form View Settings Window Help
Widget Box
Filter
Spin Box
Double Spin Box
Time Edit
Date Edit
Date/Time Edit
Dial
Horizontal Scroll Bar
Vertical Scroll Bar
Horizontal Slider
Vertical Slider
Key Sequence Edit
QsciScintilla
Display Widgets
Label
Text Browser
Graphics View
Calendar Widget
LCD Number
Progress Bar
Horizontal Line
Vertical Line
OpenGL Widget
QQuickWidget
Scratchpad
gridLayoutWidget
Dialog - untitled*
7 8 9 /
4 5 6 *
1 2 3 -
C 0 = +
Object Inspector
Object Class
b_c QPushButton
b_div QPushButton
b_eq QPushButton
b_mul QPushButton
b_sub QPushButton
b_sum QPushButton
textBrowser QTextBrowser
Property Editor
Filter
b_c : QPushButton
Property Value
geometry [(20, 190), 30 x 30]
X 20
Y 190
Width 30
Height 30
sizePolicy [Minimum, Fixed, 0, 0]
Action Editor
Filter
Name Used Text Shortcut Checkable
Signal/Slot Editor
Action Editor
Resource Browser

Desenhar a GUI

- Mudar o nome do ”Dialog” para “Calculadora”
- Ajustar o tamanho da janela de forma a que os botões e a caixa de texto fiquem centradas.
- Anotar a “*width*” e a “*height*” que aparecem na *geometry* da janela. Inserir esses valores em *minimumSize* e *maximumSize* (isto impede que o utilizador mude o tamanho da calculadora).
- Guardar o ficheiro como calculator.ui no sistema de ficheiros.

Qt Designer

File Edit Form View Settings Window Help

Widget Box

Filter

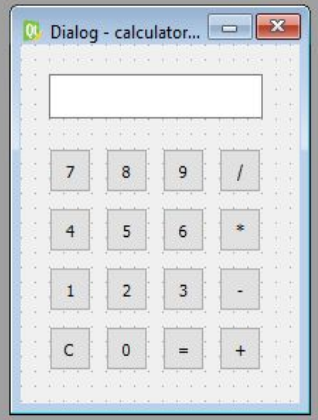
- Spin Box
- Double Spin Box
- Time Edit
- Date Edit
- Date/Time Edit
- Dial
- Horizontal Scroll Bar
- Vertical Scroll Bar
- Horizontal Slider
- Vertical Slider
- Key Sequence Edit
- QsciScintilla

Display Widgets

- Label
- Text Browser
- Graphics View
- Calendar Widget
- LCD Number
- Progress Bar
- Horizontal Line
- Vertical Line
- OpenGL Widget
- QQuickWidget

Scratchpad

- gridLayoutWidget



Object Inspector

Object	Class
Calculadora	QDialog
b_0	QPushButton
b_1	QPushButton
b_2	QPushButton
b_3	QPushButton
b_4	QPushButton
b_5	QPushButton

Property Editor

Filter

Calculadora : QDialog

Property	Value
Y	0
Width	197
Height	242
sizePolicy	[Preferred, Preferred, 0, 0]
minimumSize	197 x 242
Width	197
Height	242
maximumSize	197 x 242
Width	197
Height	242
sizeIncrement	0 x 0

Action Editor

Filter

Name	Used	Text	Shortcut	Checkable

Signal/Slot Editor | Action Editor | Resource Browser

Gerar código Python

- Abrir a linha de comandos na localização do ficheiro calculator.ui
- Correr o seguinte comando:
  - pyuic5 -x calculator.ui -o calculator.py
- Se tudo tiver corrido bem, um script python deverá ter sido gerado. Experimentar correr como nos exercícios anteriores. A calculadora desenhada deverá aparecer no ecrã. Só que os botões não fazem nada. Ainda temos de os programar!
- Possíveis problemas:
  - PyQt5 não instalado
  - PyQt5 não adicionado às variáveis de ambiente

Programar a calculadora

- Para associar uma função ao evento de clicar num botão, precisamos de, na função setupUi() gerada pelo PyQt, chamar a função connect() da variável *clicked* do botão, dando-lhe como argumento a função que queremos executar quando o botão é clicado.

```
6   #
7   # WARNING! All changes made in this file will be lost!
8
9   from PyQt5 import QtCore, QtGui, QtWidgets
10
11  class Ui_Dialog(object):
12
13      def action(self):
14              print("Botao clicado!")
```

```
176             self.b_4.setObjectName("b_4")
177             self.verticalLayout.addWidget(self.b_4)
178
179             self.button.clicked.connect(self.action)
```

- Para alterar o texto de um **TextBrowser**, basta usar a função *setText(texto)*. Para obter o seu texto, basta usar a função *toPlainText()*, que retorna o texto numa string.

Programar a calculadora (dicas)

- Posto isto, é preciso associar 16 funções aos 16 botões.
  - Os botões numéricos adicionam o seu número ao **TextBrowser**.
  - Os botões com as 4 operações registam a operação que o utilizador quer fazer.
    - Quando uma operação é escolhida ou cálculo é feito, o número que está atualmente no **TextBrowser** tem de dar lugar ao segundo número quando o utilizador premir outro dígito.
  - O botão “=” verifica a operação que o utilizador escolheu, bem como o número inicial e realiza a operação com o número que está no visor.
  - O botao “C” realiza um “reset” à calculadora.

Outras considerações

Tópicos avançados

- Estes são alguns tópicos que por motivos de tempo não foram abordados neste *workshop*, mas que para quem estiver interessado em aprofundar os conhecimentos aqui adquiridos, são uma boa base para começar:
  - Manipulação de ficheiros
  - Herança de classes
  - *List comprehensions*
  - Tuplos e sets
  - *Lambda expressions* (tornariam o desenvolvimento da lógica da calculadora consideravelmente mais rápido)
  - Iteradores (abordados de forma superficial)
  - GUI (de uma forma mais aprofundada, nomeadamente *layouts*). PyQt, TKinter
  - Web Development (com Flask ou Django, por exemplo). Requer conhecimentos de WebDev.

Mentalidade

- Fatores que tornam a aprendizagem e consolidação da programação consideravelmente mais fácil:
  - Paixão
  - Prática
  - Pesquisar quando aparece algo que não se sabe resolver.
  - Quando se pesquisa como fazer alguma coisa, não pesquisar apenas para resolver o problema na hora, mas tentar memorizar para ocasiões futuras.

Obrigado!