VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF COMPUTER SCIENCE

User Interfaces

460-2017

Spring 2024

Last update 3. 4. 2024

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

460-2017 User Interfaces

Lecturer: Tomas Fabian (tomas.fabian@vsb.cz)

- Room EA408, building of FEECS
- Office hours: Tuesday 13:30 – 15:30

  All other office hours are by appointment

Other supporting materials can be found here

http://mrl.cs.vsb.cz/people/fabian/uro_course.html

460-2017 User Interfaces

- Grading Policy
  - Three self-made projects
    - **P1**: GUI in **Python (Tk)** [30p, min 15p]; deadline 20. 3. 2024
    - **P2**: Web page with **Bootstrap** [10p, min 5p]; deadline 3. 4. 2024
    - **P3**: GUI in **C++ (Qt)** [30p, min 15p]; deadline 15. 5. 2024
  - Complete all assignments from labs
    - small **assignments from each lab** [15p, min 0p]; deadline 15. 5. 2024
  - Final exam
    - Written **test** during the last week of the semester [15p, min 5p]; deadline 15. 5. 2024

User Interfaces

- This course will give an overview and an introduction to *practices* and several *libraries* that are commonly used for *creating UIs*:
  - Define the appearance of the UI
  - Implement self-designed UI in various languages
    - Python (Tk), C++ (Qt)

Tkinter 8.6 reference: a GUI for Python

User Interfaces

- Reading assignment for the 2nd lab:

  An Introduction to Python

  https://docs.python.org/3/tutorial/introduction.html

  Graphical User Interfaces with Tk

  https://docs.python.org/3/library/tkinter.html

User Interfaces

- Reading assignment for the 2nd lab:

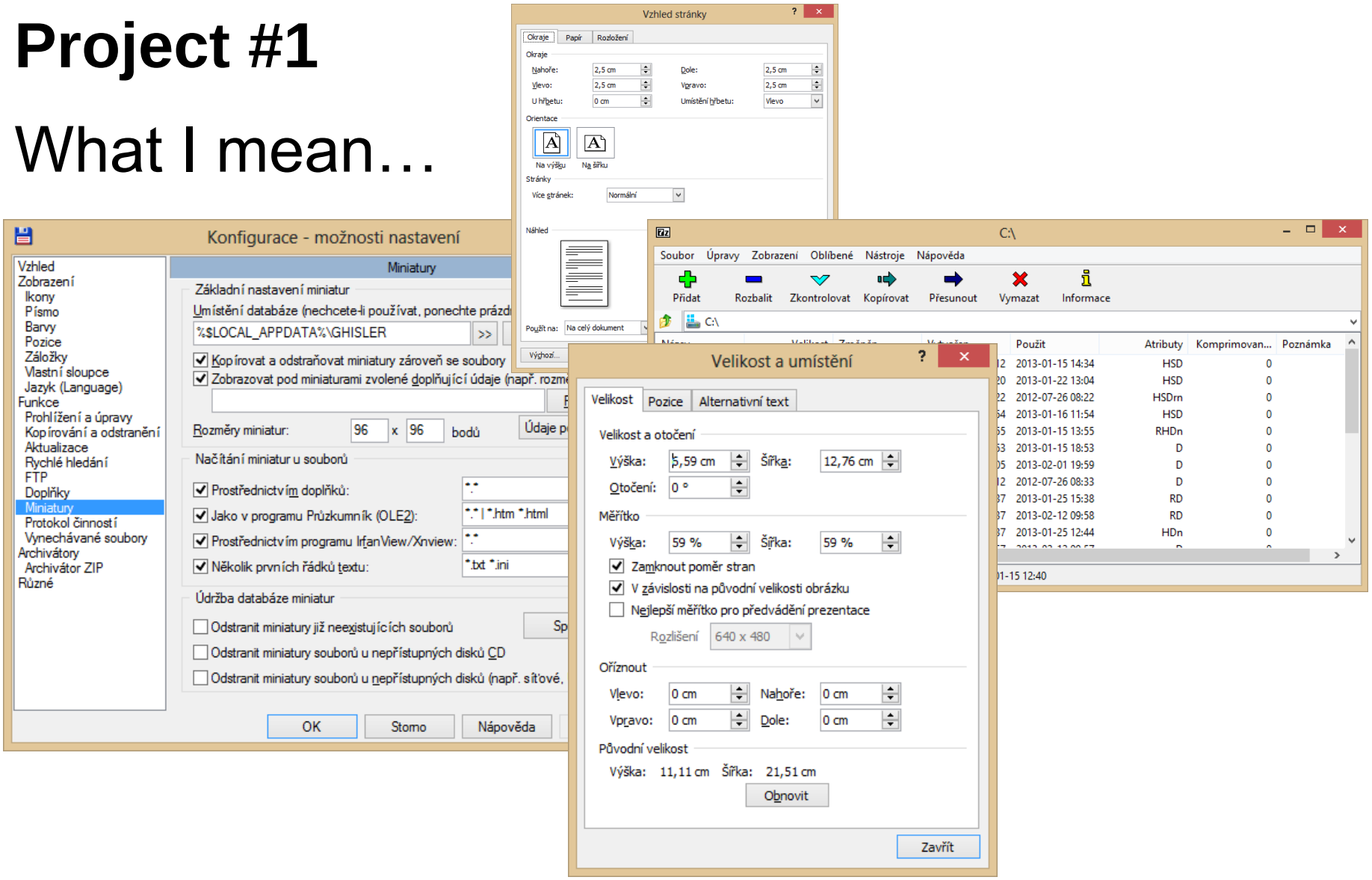
  See http://mrl.cs.vsb.cz/people/fabian/uro/links.txt
- Overview of the User Interface Development Process
- Designing a User Interface

User Interfaces

- **Project #1**
- The list of individual assignments is on the website
- Functionality is not required, only the GUI
- **Deadline is TBA (by email)**

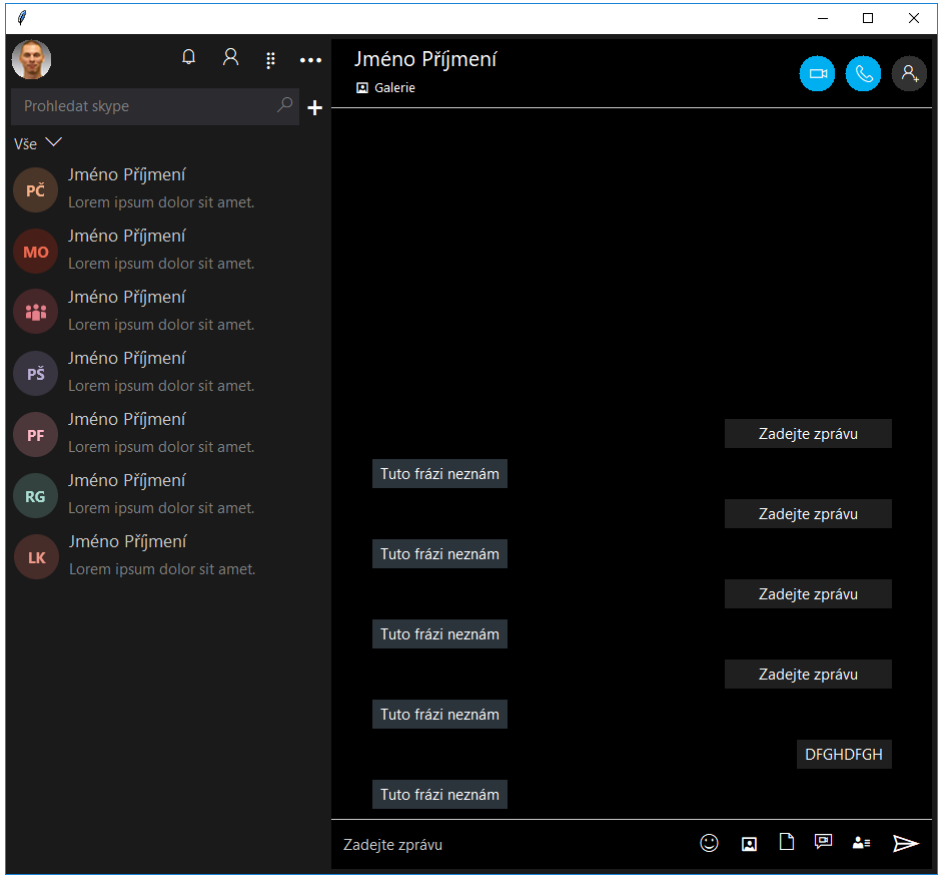
User Interfaces

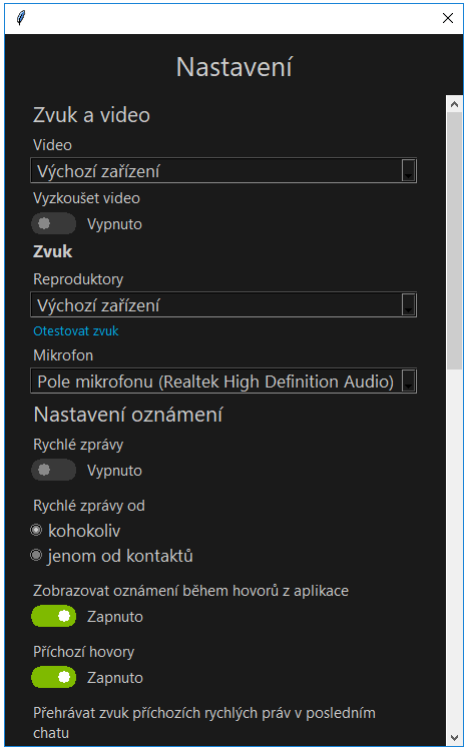
- **Project #1**
- What I mean…

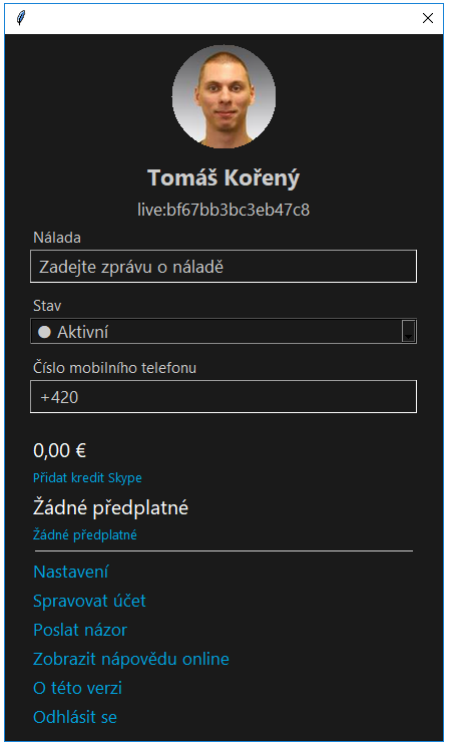


User Interfaces

- **Project #1 Example**







Project #1 – Skype, Tomáš Kořený, 2018

Designing User Interface

- Functional Requirements
  - Follow UI design guidelines (exceptions are allowed)
  - Ensure that the UI is accessible
  - Support internationalization

Source: https://msdn.microsoft.com/en-us/library/windows/desktop/ff728820%28v=vs.85%29.aspx

Designing User Interface

- User Analysis
  - Who are our users? What skills and knowledge do they have?
  - What different sources of data can we use to understand their experience?
  - What goals and tasks will they use our product to complete?
  - What assumptions are we making and how can we verify them?
  - What sources of data do we have? (Usability studies and heuristic evaluations are good places to start.)

Source: https://msdn.microsoft.com/en-us/library/windows/desktop/ff728820%28v=vs.85%29.aspx

Designing User Interface

- Conceptual Design
  - Typically, the UI is not addressed in this phase
  - This phase does require a thorough business model with complete user profiles and usage scenarios which are imperative for a successful user experience.

Source: https://msdn.microsoft.com/en-us/library/windows/desktop/ff728820%28v=vs.85%29.aspx

Designing User Interface

- Logical Design
  - The logical design phase is when the initial prototypes that support the conceptual design are developed.
  - The specific hardware and software technologies to be used during development are also identified in this phase, which can determine the capabilities of the UI in the final product.
  - In addition to the development tools, the various hardware requirements and form factors that are to be targeted by the application should be identified.

Source: https://msdn.microsoft.com/en-us/library/windows/desktop/ff728820%28v=vs.85%29.aspx

Designing User Interface

- Physical Design
  - The physical design phase determines how a UI design is to be implemented for the specific hardware and form factors that were identified in the logical design.
  - It is during this phase that hardware or form factor limitations might introduce unexpected constraints on the UI that require significant refinements to the design.

Source: https://msdn.microsoft.com/en-us/library/windows/desktop/ff728820%28v=vs.85%29.aspx

User Interfaces

- Python download

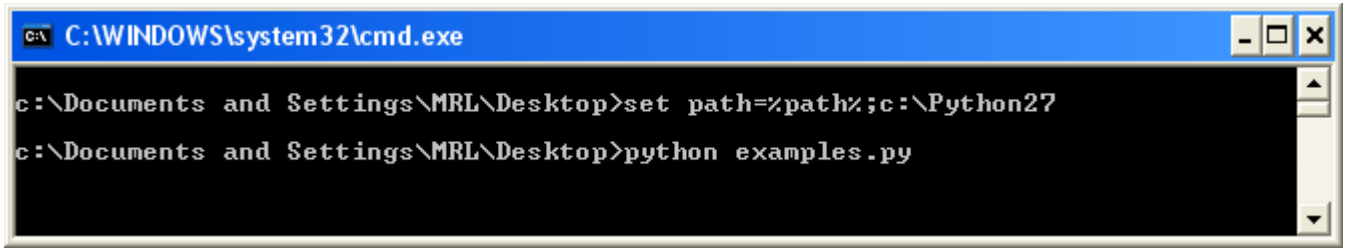
  http://www.python.org/download/
- Documentation

  http://docs.python.org/3/

  https://docs.python.org/3/library/tk.html

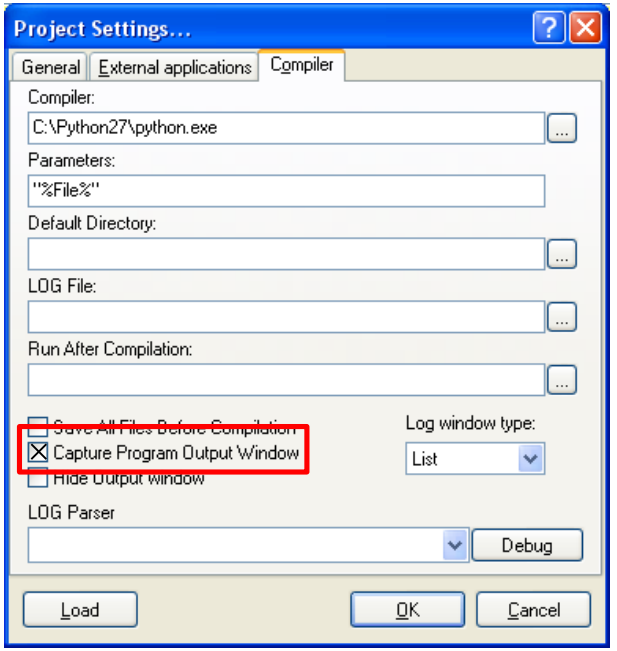
Running Python

- From command line



- From PSPad

Follow the screen-shot and then press Ctrl+F9



Python Crash Course

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

__author__    = "Tomas Fabian"
__copyright__ = "(c)2021 VSB-TUO, FEECS, Dept. of Computer Science"
__email__     = "tomas.fabian@vsb.cz"
__version__   = "0.1.0"

"""
A very simple script done in Python
"""
def main():
  print("Hello, World!")

if __name__ == "__main__":
  main()
```

Python uses indentation to define a block of code
The amount of indentation (e.g. 4 whitespaces) must be consistent throughout that block

```
c:\Users\Tomas\Documents\vsb\vyuka\ura\src>python ex01.py
Hello, World!
```

set PATH=%PATH%;"c:\Program Files\Python310"

Python Crash Course

```
# elementary commands
a = 5 # ints
a = 3.14 # floats
# strings
a = "This is 'positive'"
a = 5//3 # a equals to 1
a = 5/3 # a equals to 1.667
a = True and False
a = True or False
a = True is True
a = True == True
a = not True
# bitwise and
a = 1 & 3 # a equals to 1
# bitwise or
a = 1 | 3 # a equals to 3
```

```
# definition of a function
def my_function(a, b=0):
  c = a + b
  return c

# function call
print(my_function(1, 3))

# program flow control
a = 5
if a > 0:
  print("positive")
elif a == 0:
  print("zero")
else:
  print("negative")
# there is no switch statement
until Python 3.10
```

Python Crash Course

```
# match-case statement
http_code = "418"

match http_code:
    case "200":
        print("OK")
        do_something_good()
    case "404":
        print("Not Found")
        do_something_bad()
    case "418":
        print("I'm a teapot")
        make_coffee()
    case _:
        print("Code not found")
```

```
#  same logic using chunk of if-elif-else

if http_code == "200":
    print("OK")
    do_something_good()
elif http_code == "404":
    print("Not Found")
    do_something_bad()
elif http_code == "418"
    print("I'm a teapot")
    make_coffee()
else:
    print("Code not found")
```

Python Crash Course

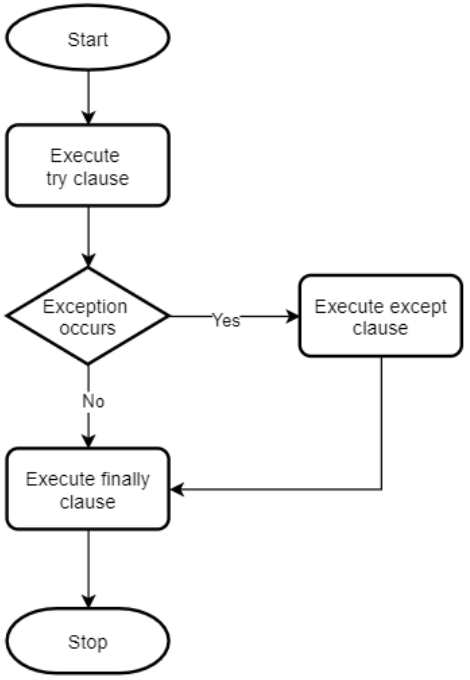
```
# abstract data types in Python
# list
lst = [1, 2, 3, "hi there"]
len(lst)
output: 4
print(type(lst[3]))
output: <class 'str'>
lst[1] = 22 # list is mutable
print(lst)
output: [1, 22, 3, "hi there"]

# tuple
vec = (1.2, 3.4, -8.9)
print(len(vec))
output: 3
vec[0] = -1.2 # tuple is immutable
```

```
# dict - dictionary (mutable)
dct = {1: "one", 2: "two"}
print(dct[2])
output: two

# set (immutable)
s = set((1, 2, 3, 4, 4, 4))
print(s)
output: {1, 2, 3, 4}
```

Try `help(list)` command to see what all you can do with ADTs

Python Crash Course

```
# list comprehension
lst = [1, 2, 3, 4, '3.14']
lst = [x + 1 for x in lst if type(x) in [int, float]]
output: [2, 3, 4, 5]

# exception handling
for x in lst:
  try:
    x += 1
  except TypeError:
    print('x is not a number')
  finally:
    print(x)
```



Python Crash Course

```
# parent class
class BaseWindow():
  max_width = 3840 # class variable shared by all instances

  def __init__(self, width=640): # constructor
    self.width = min(width, BaseWindow.max_width) # instance variable

  def resize(self, width): # method
    self.width = width
```

Python Crash Course

```
# derived class
class MyWindow(BaseWindow): # Python supports multiple inheritance
  def __init__(self, title, width, height):
    super().__init__(width) # calls constructor of the parent class
    self.title = title
    self.height = height

  def __str__(self): # to string method
    return "Window '{}' has {}x{} pixels".format(self.title,
self.width, self.height)


# instantiate an object of type MyWindow
window = MyWindow("Example", 800, 600)
print(window)
output: Window 'Example' has 800x600 pixels
```

Python Crash Course

```
# anonymous function
fce = lambda a, b: a + b
print(fce(1, 3))
output: 4
```

Note that lambda functions are callable objects and they can be invoked using the call operator

GUI in Python (Tk)

- We will use Tkinter for constructing GUIs in the Python
- Tkinter is standard GUI toolkit for Python
- Object-oriented layer on top of Tcl/Tk
- Pertinent references:
  - http://effbot.org/tkinterbook/
  - https://docs.python.org/3/library/tk.html

GUI in Python (Tk)

- List of common widgets:

- Label
- Button
- Entry
- Spinbox
- Checkbutton
- Radiobutton
- Listbox
- Text
- Message
- Scale
- Scrollbar
- Canvas
- Frame
- LabelFrame
- Toplevel
- PanedWindow
- Menu
- Menubutton
- Modules:
- tkMessageBox
- tkFont

GUI in Python (Tk)

```
# -*- coding: utf-8 -*-

import tkinter as tk                    #imports the entire Tk package
#from tkinter import Tk, Label
#from tkinter import *

root = tk.Tk()
label = tk.Label(root, text="Some text")
label.pack()
root.mainloop() # Starts the app's main loop events
```

GUI in Python (Tk)

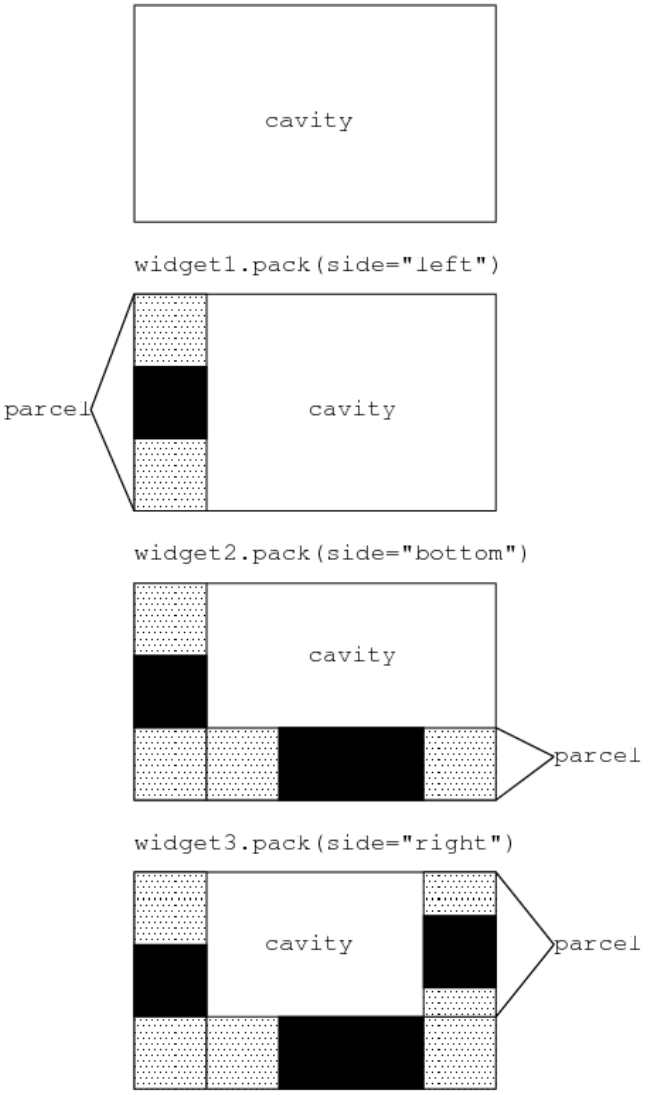
```
import tkinter as tk      #imports the entire Tk package

class Application(tk.Frame):    # App class inherits from Frame class
  def __init__(self, master=None):
    tk.Frame.__init__(self, master) # Calls constructor for the parent class
    self.pack()                     # Make the app appear on the screen
    self.createWidgets()

  def createWidgets(self):  # Function responsible for creating all the widgets
    self.hi_there = tk.Button(self) # Creates the button
    self.hi_there["text"] = "Hello World\n(click me)" # Set button's parameters
    self.hi_there["command"] = self.say_hi
    self.hi_there.pack(side="top") # Places the button

    self.QUIT = tk.Button(self, text="QUIT", fg="red", command=root.destroy)
    self.QUIT.pack(side="bottom")

  def say_hi(self):
    print("hi there, everyone!")

root = tk.Tk()
app = Application(master=root)          # Instantiating the App class
app.master.title("Sample application")  # Sets the title of the window
app.mainloop() # Starts the app's main loop; waiting for mouse and keyboard
events
```

GUI in Python (Tk)

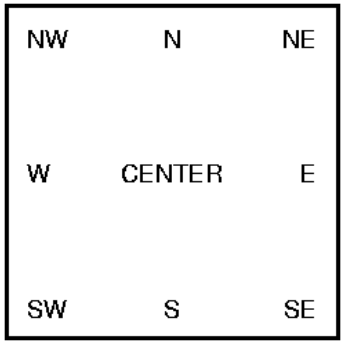
- *Window* – **rectangular area** somewhere on the screen
- *Top-level window* – a window that exists **independently** on the screen
- *Widget* – generic therm for any **building block** that make up a GUI
- e.g. Button, Label, Entry, Frame
- *Frame* – basic widget that can **contain** other widgets. You can create **complex layouts** with them.

Layout management

- Widgets are arranged in a window by three different geometry managers
- PACK .pack()



Layout management



- Widgets are arranged in a window by three different geometry managers
- **PACK** .pack()
  - **expand** – 1 widget will be expanded to fill any empty space, 0 otherwise
  - **anchor** – specifies how the widget is placed inside its parcel, see the compass

    ‘n’, ‘ne’, ‘e’, ‘se’, ‘s’, ‘sw’, ‘w’, ‘nw’, and ‘center’
  - **pad**{**x**,**y**} – designating external padding on each side of the slave widget

Layout management

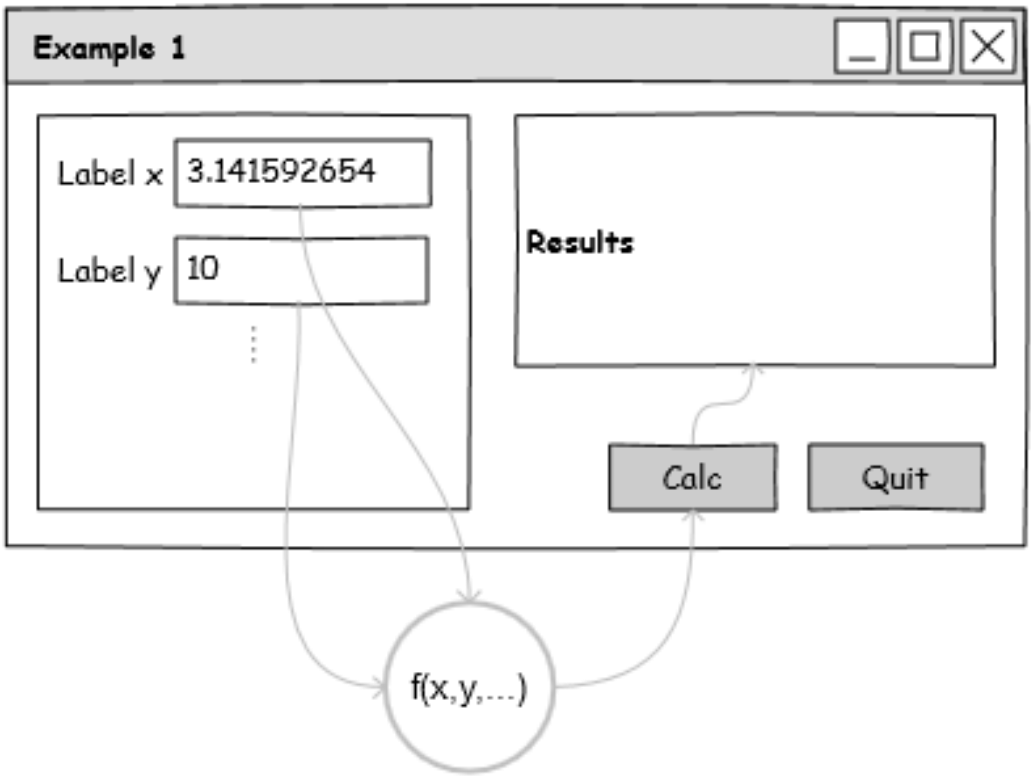
- Widgets are arranged in a window by three different geometry managers
- **PACK** .pack()
  - **fill** – fill any empty space in ‘x ’, ’ y ’, ’ both ’, ’ none ’
  - **ipad{x,y}** – designating internal padding on each side of the slave widget
  - **side –** ‘left’, ‘right’, ‘top’, ‘buttom’
  - `self.widget = Constructor(parent, ...)`
  - `self.widget.pack(...)`

Layout management

- Widgets are arranged in a window by three different geometry managers
- **PACK** .pack()
  - More on **expand**:
  - Expand specifies whether the widgets should be expanded to fill any extra space in the geometry master. If false (default), the widget is not expanded.
  - The expand option tells the manager to assign additional space to the widget box. If the parent widget is made larger than necessary to hold all packed widgets, any exceeding space will be distributed among all widgets that have the expand option set to a non-zero value.

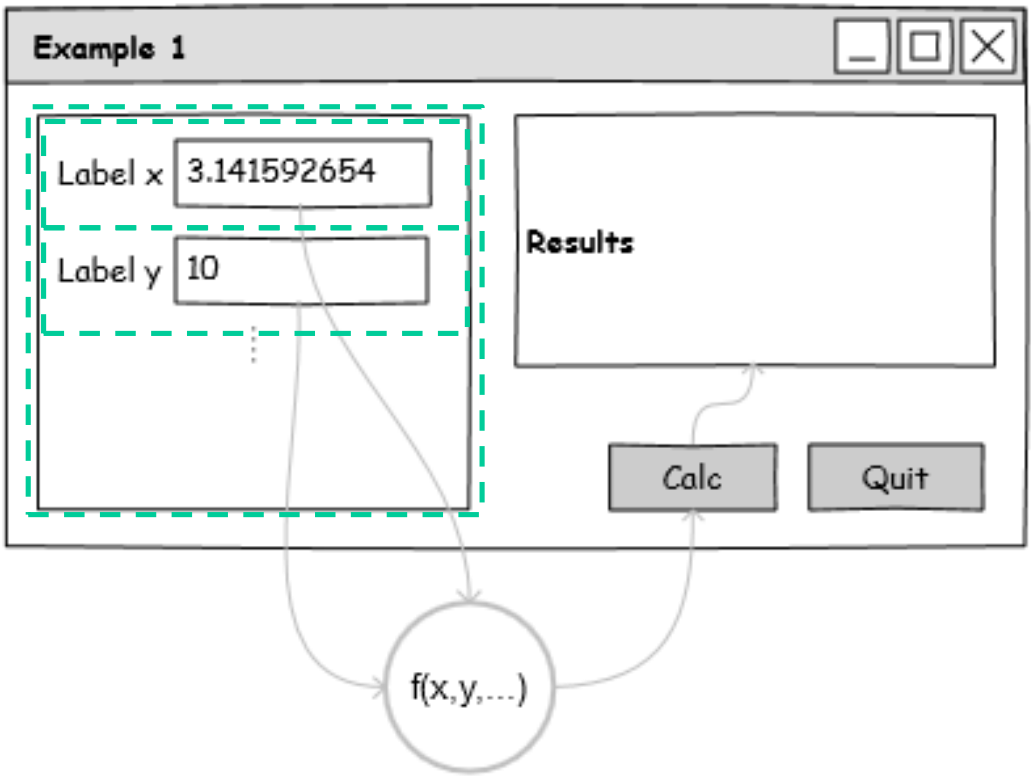
Example 1

- Arrange the widgets (Frame, Label, Button, Entry) according the following mockup and realize some calculation with typed numbers:



Example 1

- Arrange the widgets (Frame, Label, Button, Entry) according the following mockup and realize some calculation with typed numbers:



Accessing Entries

- self.x = tkinter.StringVar()
- self.ent_value_x = tkinter.Entry(self, **textvariable = self.x**, justify = tkinter.RIGHT)
- self.ent_value_x.insert(0, "0.0") # set new string value directly to entry field
- #self.ent_value_x.delete(0, tkinter.END) # delete all characters
- #**self.x.set**("10.546") #set value to entry field via text variable x
- value = float(**self.x.get**()) # type check needed, see the next slide

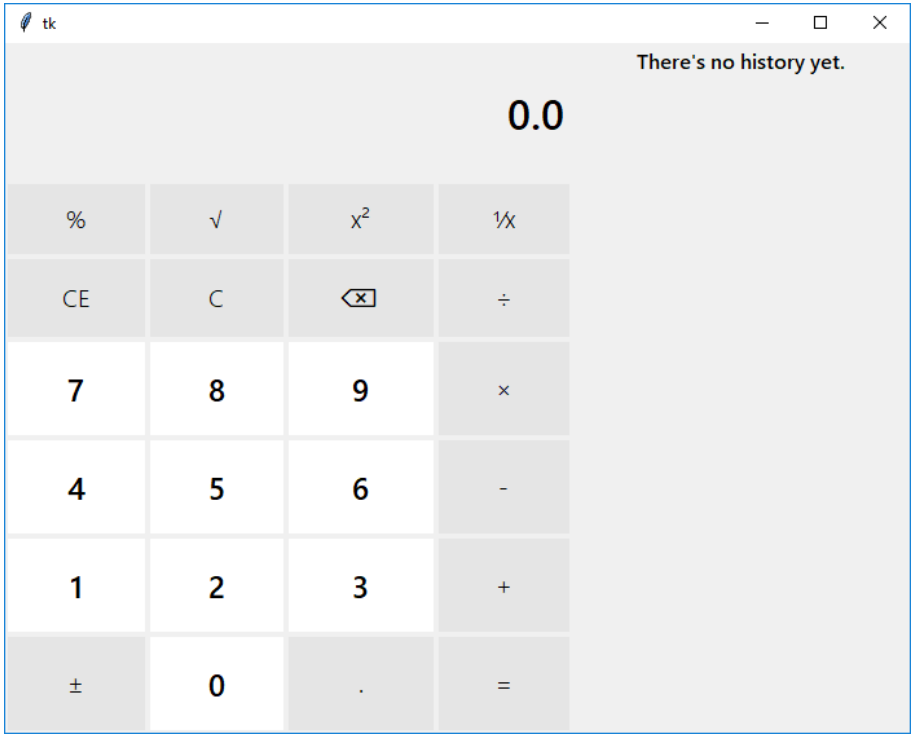
Type Validation

- import tkinter.messagebox as mb

- **try**:
- 	x = float(self.x.get())
- 	print("x=" + str(x))
- **except** ValueError:
- 	mb.showwarning("Value Error", "Real number required.")
- 	self.x.set("")

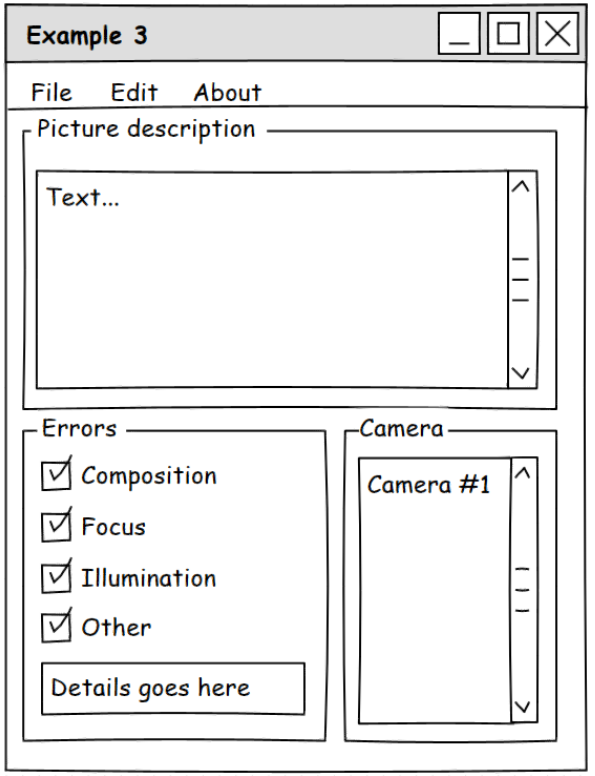
Example 2

- Arrange the widgets to represent the layout of common calculator (Frame, Label, Button, Entry, use the Grid layout manager) according the following screen-shot and realize some calculation:



Example 3

- Arrange the widgets to represent the layout of image labeler (use LabelFrame, Menu, Listbox, Scrollbar, Checkbox, Entry, File dialog) according the following mockup:

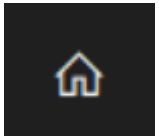


Example 3

```
btn = tk.Button(root, text=label, font=fontNumPads if label.isdigit() else fontPads, relief=tk.FLAT, bg="white" if label.isdigit() else "gray90", command=lambda number = label : addNumber(number))

# change the background color whent the mouse enter the widget

btn.bind("<Enter>", lambda event: event.widget.configure(bg="gray80"))

# and whent the mouse leave the widget

btn.bind("<Leave>", lambda event: event.widget.configure(bg="white" if event.widget["text"].isdigit() else "gray90"))
```

Example 3

```
# setup the right font for Windows 10 like apps

fontText = tkf.Font(family='Segoe UI Semibold', size=12,
weight='normal')

# icons are done with the aim of fonts as well

icons = tkf.Font(family='Segoe MDL2 Assets', size=13,
weight='normal')
```



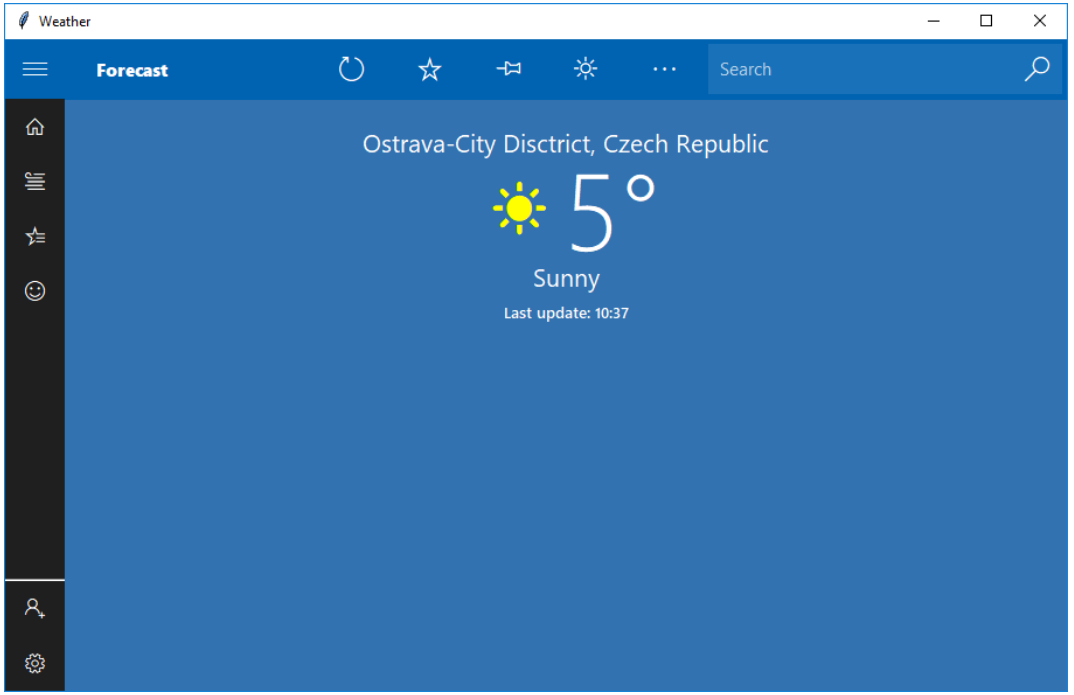
```
lblHouse = tk.Label(frmLeft, text="\ue80f", font=icons,
bg=gray, fg=white)
```

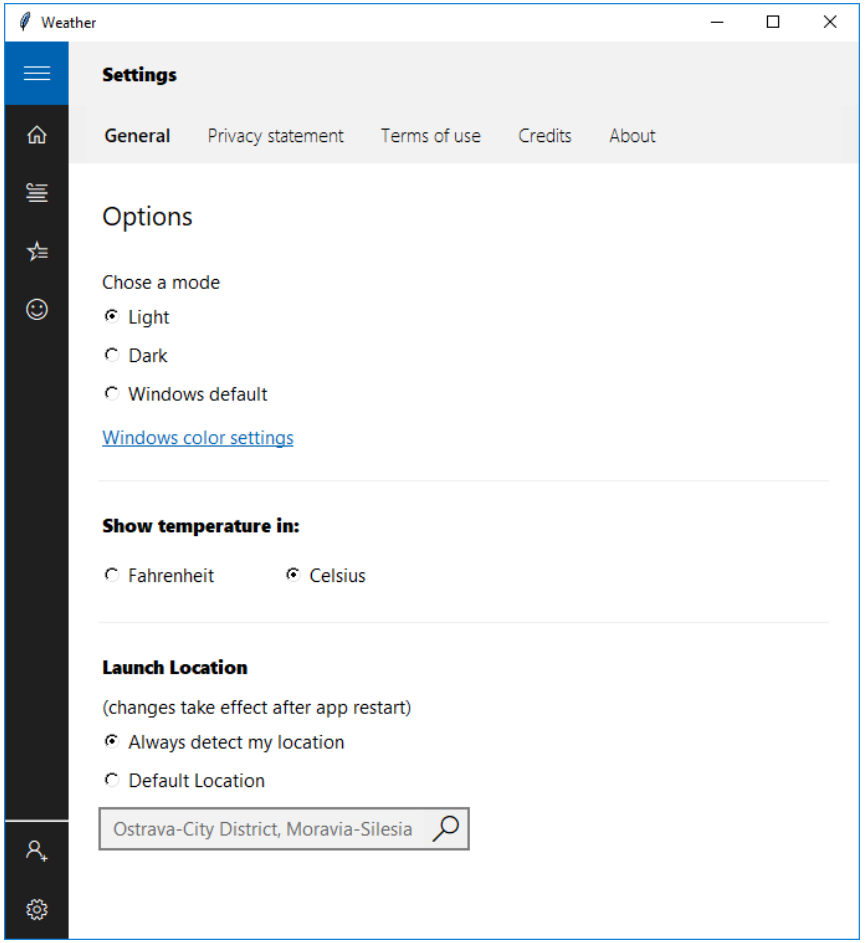
See the Icons list for further reference…

https://docs.microsoft.com/en-us/windows/uwp/design/style/segoe-ui-symbol-font#icon-list

Example 4

- Arrange the widgets to represent the layout of Weather app from Windows Store





Layout management

- Widgets are arranged in a window by three different geometry managers
- **GRID** .grid()
  - Threats every windows or frame as a table
  - If widget do not fill entire cell, you can specify what happens to the extra space (leave the extra space outside the widget or stretch the widget to fit it)
  - Spanning – combine cells into one larger area
  - All widgets have a .grid() method
  - `self.widget = Constructor(parent, ...)`
  - `self.widget.grid(...)`

Layout management - GRID

- `widget.grid(option=value, …)`
- column, row – cell position, counting from 0
- columnspan, rowspan – merge multiple cells into one larger cell
- ipad{x,y} – internal padding, dimension is added inside the widget inside its {left and right, top and bottom} borders
- Sticky – default is to center the widget in the cell

Layout management - GRID

- Sticky
  - N (top center) ,S, E, W
  - NE (top right), SE, SW, NW
  - N+S (stretched vertically and centered horizontally)
  - E+W (stretched horizontally and centered vert.)
  - N+S+W (stretch the widget vertically and place it left)
  - N+E+S+W (stretch the widget to fill the cell)

Layout management - GRID

- w.grid_bbox(column=0, row=0[, col2=1, row2=1]) – returns 4-tuple describing bounding box
- w.grid_forget() - w disappear from the screen
- w.grid_remove() - same as forget, but grid options are remembered
- w.grid_info() - returns values of w's options
- w.grid_propagate() - force a widget to be a certain size, regardless of the size of its contents

Layout management - GRID

- w.grid_slaves(row=None, column=None) – returns a list of the widgets managed by w
- w.grid_size() - return 2-tuple containing number of columns and rows
- w.columnconfigure(N, option=value, …)
- w.rowconfigure(N, option=value, …)

  Options = {minsize, pad, weight}

```
w.columnconfigure(0, weight=3)
w.columnconfigure(1, weight=1)
```

  It will distribute three-fourths of the extra space to the first column and one-fourth to the second column

Layout management - GRID

- Let the user resize your entire application window, and distribute the extra space among its internal widgets.
- `top = self.winfo_toplevel() # get the top-level window`
- `top.rowconfigure(0, weight=1) # makes row 0 stretchable`
- `top.columnconfigure(0, weight=1) # makes column 0 st.`
- `self.rowconfigure(0, weight=1) # same for App widget`
- `self.columnconfigure(0, weight=1)`
- `self.grid(row=0, column=0, sticky=N+S+E+W) # the app widget will expand to fill its cell of the top-level window's grid`

Layouting Widgets in Loop

- **self.reds = []**

- for i in range(30):
- red = tk.IntVar(root)
- red.set(255)
- tk.Spinbox(frame, from_=0, to=255, justify="right", textvariable=red, width=3).pack(side="left")
- **self.reds.append(red)**
- **red.trace("w", lambda name, index, mode, id = i : self.setColor(id))**
- tk.Button(frame, image=self.icoPen, text=str(i), **command=lambda id = i : self.setColor(id)**).pack(side="left")

- def setColor(self, id):
- print(str(id) + " => " + str(self.reds[id].get()))

Toolbar

- `self.toolbar = tk.Frame(self, bd=1, relief=tk.RAISED)`
- `self.toolbar.pack(side=tk.TOP, fill=tk.X)`
- `self.icoQuit = tk.PhotoImage(file="quit.png")`
- `self.quitButton = tk.Button(self.toolbar, image=self.icoQuit, relief=tk.FLAT, command=self.quit, text="X")`
- `self.quitButton.pack(side=tk.LEFT, padx=2, pady=2)`

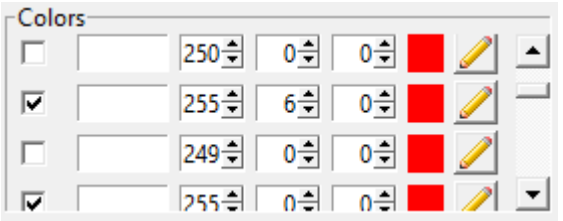
Status bar

- `self.frameStatus = tk.Frame(self, relief="sunken", border=3)`
- `self.frameStatus.pack(side="bottom", fill="x", expand=0, padx=1, pady=1)`
- `self.status = tk.Label(self.frameStatus, text="Up and running...")`
- `self.status.pack(side="left")`

- `self.separator = tk.Frame(self, height=2, bd=1, relief=tk.SUNKEN)`
- `self.separator.pack(side="bottom", fill=tk.X, padx=5, pady=5)`

Scrollable Frame 1/2

```
self.frameColor = tk.LabelFrame(self.frameleft, text="Colors",
relief="sunken", border=1)
self.frameColor.pack(side="top", fill="both", expand=1)

self.canvasColor = tk.Canvas(self.frameColor)
self.canvasColor.pack(side="left", fill="both", expand=1)

self.scrollbarColor = tk.Scrollbar(self.frameColor, orient="vertical",
command=self.canvasColor.yview)
self.scrollbarColor.pack(side="left", fill="y")
self.canvasColor.configure(yscrollcommand=self.scrollbarColor.set)

self.frameScrollableColors = tk.Frame(self.canvasColor)
self.frameScrollableColors.pack(side="top", fill="both", expand=1)
self.canvasColor.create_window((0,0),window=self.frameScrollableColors
,anchor='nw')

self.frameScrollableColors.bind("<Configure>",self.myfunction)
```



Scrollable Frame 2/2

```
def myfunction(self, event):
    self.canvasColor.configure(scrollregion =
self.canvasColor.bbox("all"),width=250,height=0)
```

Standard attributes

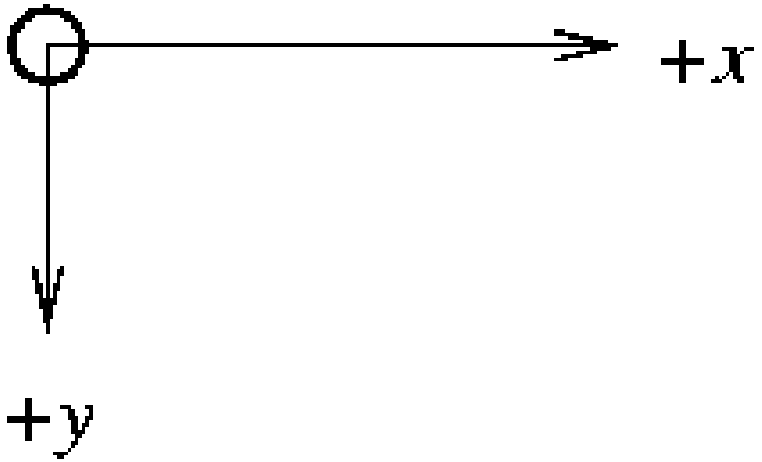
- Each widget has a set of attributes such as fonts, colors, sizes, text labels, ...
- After you have created a widget, you can later change any option by using the widget's .**config()** method.
- `self.btnD.config(width=5)`
- You can retrieve the current setting of any option by using the widget's .**cget()** method.
- `print 'width=' + str(self.btnD.cget('width'))`

Dimensions

- If you set a dimension to an **integer**, it is assumed to be in **pixels**.
- You can specify units by setting a dimension to a string containing a number followed by:
- **c** Centimeters, **i** Inches, **m** Milimeters, **p** (1/72”)
- For example, "350" means 350 pixels, "350c" means 350 centimeters, "350i" means 350 inches, and "350p" means 350 printer's points (1/72 inch).
- `w.grid(..., ipadx="5c")`

The coordinate system

- The **origin** of each coordinate system is at its **upper left corner**, with the **x** coordinate increasing toward the right, and the **y** coordinate increasing toward the bottom.



Colors

- There are two ways to specify colors:
  - You can use a string specifying the proportion of red, green, and blue in hexadecimal digits:

#rgb	Four bits per color
#rrggbb	**Eight bits per color**
#rrrgggbbb	Twelve bits per color

  - You can also use any locally defined standard color name ("white", "black", "red", "green", "blue", "cyan", "yellow", "magenta", ...).

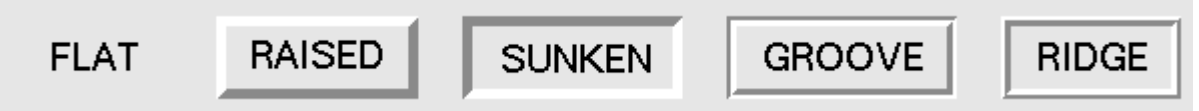
Fonts

- There two ways to specify type style.
  - As a tuple whose first element is the font family, followed by a size in points, optionally followed by a string containing one or more of the style modifiers bold, italic, underline, and overstrike.
  - ("Times", "24", "bold italic")
  - You can create a “font object” by importing the font module and using its Font class constructor.
  - `import tkinter.font as tkf`
  - `font = tkf.Font(family="Helvetica", size=36, weight="bold/normal", slant=”italic/roman”, underline=0/1, overstrike=0/1)`

`https://docs.microsoft.com/en-us/windows/uwp/design/style/segoe-ui-symbol-font`

Relief style

- The relief style of a widget refers to certain simulated 3-D effects around the outside of the widget.



- The borderwidth attribute of the widget controls width of these borders.
- `Label(..., relief=RAISED, borderwidth=2)`

Bindings and Events

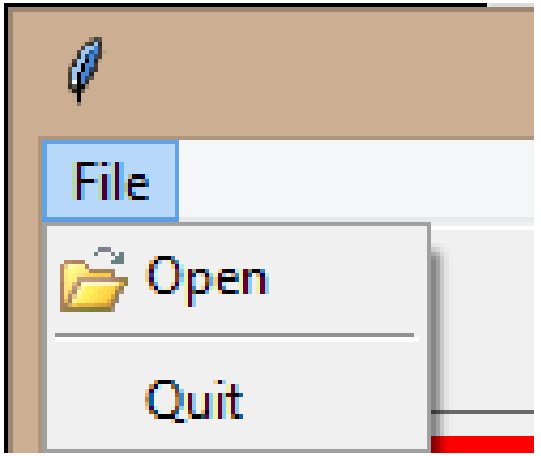
- Allows you to watch for certain events
- `widget.bind(event, handler)`
- Handler (callback) function trigger when event occurs
-
```
def click(event):
    print 'You have just clicked on
      {0}'.format(event.widget['text'])
```

- `btn = Button(text='Press me')`
- `btn.pack()`
- `btn.bind('<Button-1>', click, [add='' or '+'])`
- Add is optional, either '' or '+'. Passing an empty string denotes that this binding is to replace any other bindings that this event is associated with. Passing a '+' means that this function is to be added to the list of functions bound to this event type.

Bindings and Events

- Events are given as a string that denotes the target kind of event
- Syntax: <modifier-type-detail>
- `<Configure>`, `<ButtonPress-1>`, `<Button1-Motion>`, `<4>` (mouse wheel up), `<5>` (mouse wheel down), `<Double-1>` (double click), `<ButtonRelease-1>`, `<Shift-ButtonRelease-1>`, `<Motion>` (mouse move), `<KeyPress>`, `<<right-click>>`
- Add is optional, either '' or '+'. Passing an empty string denotes that this binding is to replace any other bindings that this event is associated with. Passing a '+' means that this function is to be added to the list of functions bound to this event type.

Menu




```
self.menubar = Menu(self.master)
self.master.config(menu=self.menubar)

self.filemenu = Menu(self.menubar, tearoff=0)
self.ico_open = PhotoImage(file="open.png")
self.filemenu.add_command(label="Open", command=self.open,
   image=self.ico_open, compound="left")
self.filemenu.add_separator()
self.filemenu.add_command(label="Quit",
command=self.master.quit)

self.editmenu = …

self.menubar.add_cascade(label="File", menu=self.filemenu)
self.menubar.add_cascade(label="Edit", menu=self.editmenu)

self.filemenu.entryconfig(0, state=DISABLED)
```

Checkbutton

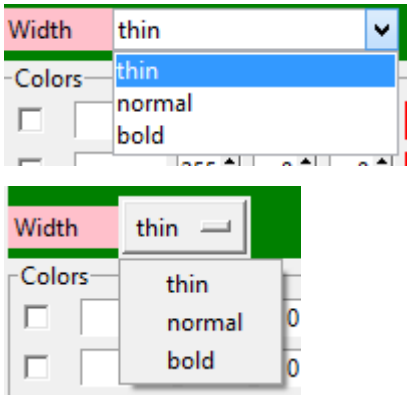
```
self.we = StringVar()
self.c1 = Checkbutton(self.master, text="Label",
variable=self.we, onvalue="bold", offvalue="normal",
command=self.method)
…
self.c1.select()
self.c1.deselect()
```

Listbox


```
self.v = StringVar()
self.v.set("Tree Grass Bush Blossom")

self.lsbOptions = Listbox(self.frmOptions,
listvariable=self.v, selectmode=SINGLE, height=1)
self.lsbOptions.pack(side="left", fill=BOTH, expand=1)
self.lsbOptions.insert(END, "Palm tree")

self.scbOptions = Scrollbar(self.frmOptions)
self.scbOptions.pack(side="left", fill="x")

self.lsbOptions.config(yscrollcommand=self.scbOptions.set)
self.scbOptions.config(command=self.lsbOptions.yview)
```

BROWSE: Normally, you can only select one line out of a listbox. If you click on an item and then drag to a different line, the selection will follow the mouse. This is the default.
SINGLE: You can only select one line, and you can't drag the mouse. Wherever you click button 1, that line is selected.
MULTIPLE: You can select any number of lines at once. Clicking on any line toggles whether or not it is selected.
EXTENDED: You can select any adjacent group of lines at once by clicking on the first line and dragging to the last line.

Combo Box




```
from tkinter import ttk
…
frame_width = tk.Frame(frameleft, bg="green")
frame_width.pack(side="top", fill="x")
label_width = tk.Label(frame_width, width=7, anchor="w",
bg="pink", text="Width")
label_width.pack(side="left", pady="5")
brush_width = tk.StringVar(root)
brush_width.set("thin") # default value
#Option menu from tk
#option_brush_width = tk.OptionMenu(frame_width,
brush_width, "thin", "normal", "bold")
#or regular Combo box from ttk
option_brush_width = ttk.Combobox(frame_width,
textvariable=self.brush_width, values=["thin", "normal",
"bold"])
option_brush_width.pack(side="left")
```

New Window

```
otherwindow = Toplevel(self)
otherwindow.resizable(width=TRUE, height=TRUE)
Otherwindow.title(“Dialog”)

other.transient(self) # this makes otherwindow modal
other.grab_set()

btn1 = Button(otherwindow, text=“Button”)
Btn1.pack()
```

The Toplevel widget works pretty much like Frame, but it is displayed in a separate, top-level window.

Message Box


```
import tkinter.messagebox
…
tkinter.messagebox.showinfo("title", "message")

{showinfo, showwarning, showerror, askquestion, askyesno,
askokcancel, askretrycancel}
```

The messagebox provides an interface to the message dialogs.

```
if (tkinter.messagebox.askyesno("Question", "Should I do
it?")):
   doit()
else:
   dontdoit()
```

File Dialog

```
from tkinter import filedialog
…
filedialog.askopenfilename(parent=self.master,
title="Selection", filetypes=[("All files", "*.*"), ("GIF
files", "*.gif")])
```

Progress Bar

```
self.progress = ttk.Progressbar(self.frameStatus,
orient=tk.HORIZONTAL, length=200, mode="indeterminate")
self.progress.pack(side="left", padx=5, pady=1)
```

Or "determinate"

Determinate:

```
self.work = tk.IntVar()
self.work.set(75)
self.progress = ttk.Progressbar(self.frameStatus,
orient=tk.HORIZONTAL, length=200, mode="determinate",
maximum=100, variable=self.work)
```

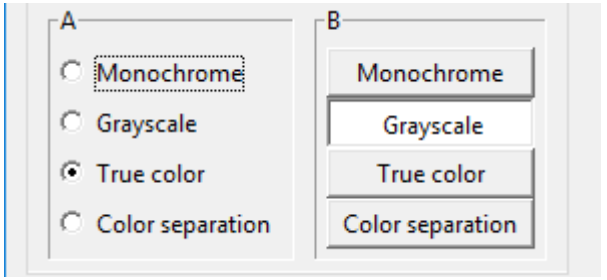
Indeterminate:

```
self.progress.start(50)
…
self.progress.stop()
```

Tabbed Panes

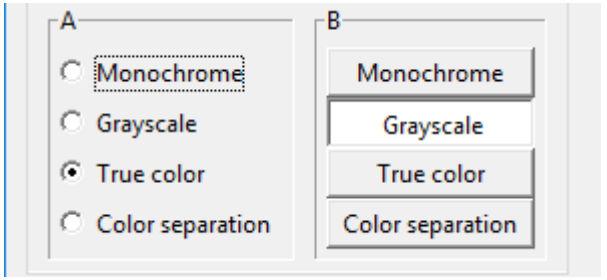

```
import tkinter as tk
from tkinter import ttk
from tkinter.scrolledtext import ScrolledText

self.options = ttk.Notebook(self.frameleft)
self.options.pack(side="top", fill="x")

page1 = tk.Frame(self.options, padx=5, pady = 5)
self.options.add(page1, text="One")
page2 = tk.Frame(self.options, padx=5, pady = 5)
self.options.add(page2, text="Two")

self.text1 = ScrolledText(page1, width = 10, height = 5)
self.text1.pack(fill="both", expand=1)
```

Radio Buttons



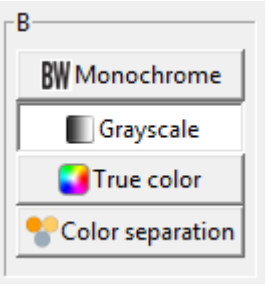

```
self.choice1 = tk.LabelFrame(page2, text="A", relief="sunken", border=1,
pady=5)
self.choice1.pack(side = "left", fill="both", expand=1, padx=5)

self.choice2 = tk.LabelFrame(page2, text="B", relief="sunken", border=1,
pady=5)
self.choice2.pack(side = "left", fill="both", expand=1, padx=5)

MODES = [ ("Monochrome", "1"), ("Grayscale", "L"), ("True color", "RGB"),
        ("Color separation", "CMYK")]

self.v1 = tk.StringVar()
self.v1.set("RGB") # initialize
for text, mode in MODES:
  rb = tk.Radiobutton(self.choice1, text=text, variable=self.v1,
value=mode).pack(anchor=tk.W)

self.v2 = tk.StringVar()
self.v2.set("L") # initialize
for text, mode in MODES:
  rb = tk.Radiobutton(self.choice2, text=text, variable=self.v2,
value=mode, indicatoron=0).pack(fill="x", padx = 5)
```

Radio Buttons



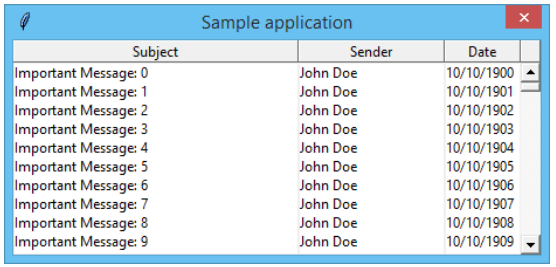

```
self.choice1 = tk.LabelFrame(page2, text="A", relief="sunken", border=1,
pady=5)
self.choice1.pack(side = "left", fill="both", expand=1, padx=5)

self.choice2 = tk.LabelFrame(page2, text="B", relief="sunken", border=1,
pady=5)
self.choice2.pack(side = "left", fill="both", expand=1, padx=5)

MODES = [ ("Monochrome", "1"), ("Grayscale", "L"), ("True color", "RGB"),
        ("Color separation", "CMYK")]

self.v1 = tk.StringVar()
self.v1.set("RGB") # initialize
for text, mode in MODES:
  tk.Radiobutton(self.choice1, text=text, variable=self.v1,
value=mode).pack(anchor=tk.W)

self.v2 = tk.StringVar()
self.v2.set("L") # initialize
for text, mode in MODES:
  tk.Radiobutton(self.choice2, text=text, variable=self.v2, value=mode,
indicatoron=0).pack(fill="x", padx = 5)
```

Radio Buttons




```
self.choice2 = tk.LabelFrame(page2, text="B", relief="sunken", border=1,
pady=5)
self.choice2.pack(side = "left", fill="both", expand=1, padx=5)

self.MODES = [
  ("Monochrome", "1", tk.PhotoImage(file="monochrome.png")),
  ("Grayscale", "L", tk.PhotoImage(file="grayscale.png")),
  ("True color", "RGB", tk.PhotoImage(file="truecolor.png")),
  ("Color separation", "CMYK", tk.PhotoImage(file="colorsep.png"))
]

self.v2 = tk.StringVar()
self.v2.set("L") # initialize

for text, mode, ico in self.MODES:
  tk.Radiobutton(self.choice2, text=text, variable=self.v2, value=mode,
indicatoron=0, image=ico, compound="left").pack(fill="x", padx = 5)
```

Tables




```
import tkinter as tk
from multilistbox import MultiListbox
…
mlb = MultiListbox(self, (('Subject', 40), ('Sender', 20), ('Date', 10)))
for i in range(1000):
  mlb.insert(tk.END, ('Important Message: %d' % i, 'John Doe',
'10/10/%04d' % (1900+i)))
mlb.pack(expand=tk.YES, fill=tk.BOTH)
```

Download the multilistbox.py file from my web site and copy it into your project's folder.

List of Ttk Widgets

- Ttk comes with 17 widgets, 11 of which already exist in Tkinter: Button, Checkbutton, Entry, Frame, Label, LabelFrame, Menubutton, PanedWindow, Radiobutton, Scale, and Scrollbar
- The 6 new widget classes are: Combobox, Notebook, Progressbar, Separator, Sizegrip, and Treeview
- All of these classes are subclasses of Widget

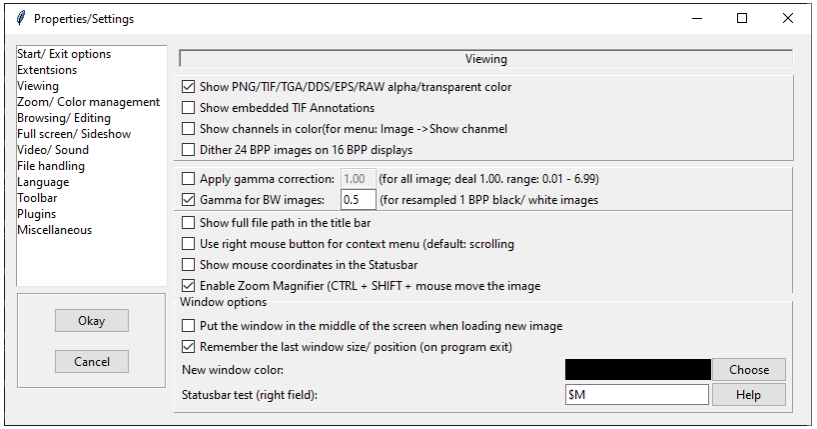
- Each widget in ttk is assigned a style:

- `from tkinter import ttk`

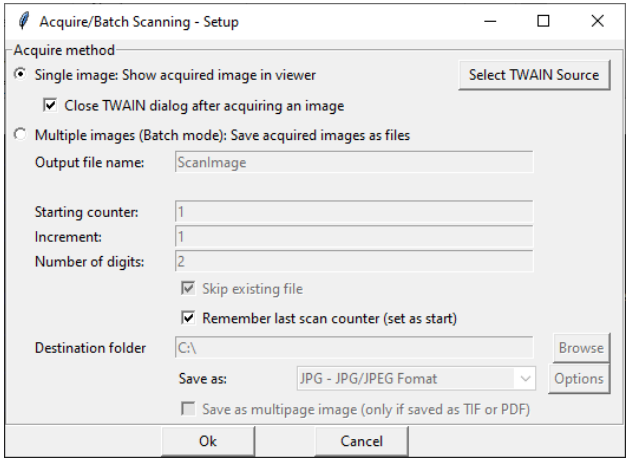
- `style = ttk.Style()`
- `style.configure("BW.TLabel", foreground="black", background="white")`
- `label = ttk.Label(text="Test", style="BW.TLabel")`

More Tk Examples

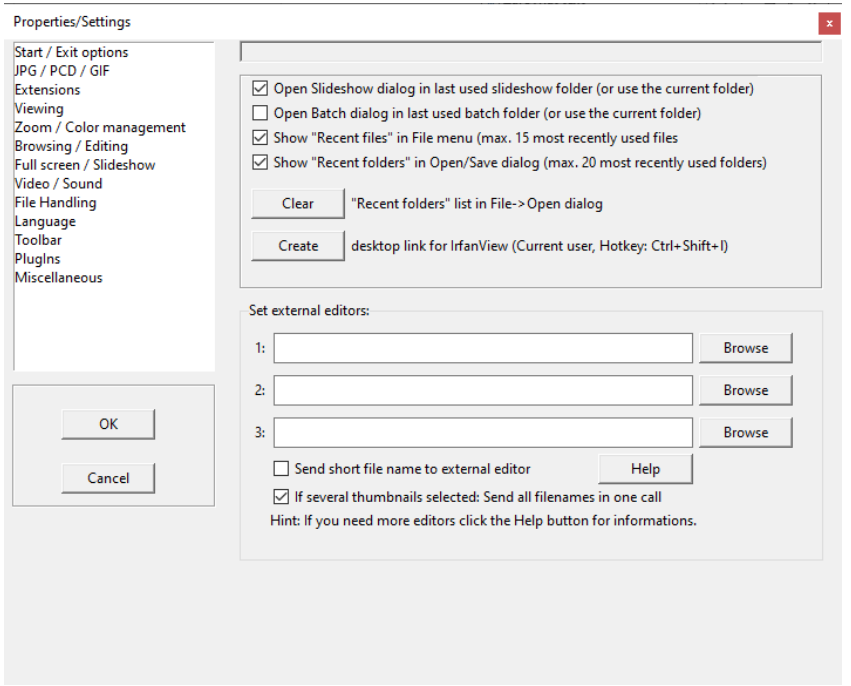
- Examples of Python/Tk implementations of more complex GUIs



tk_02.py



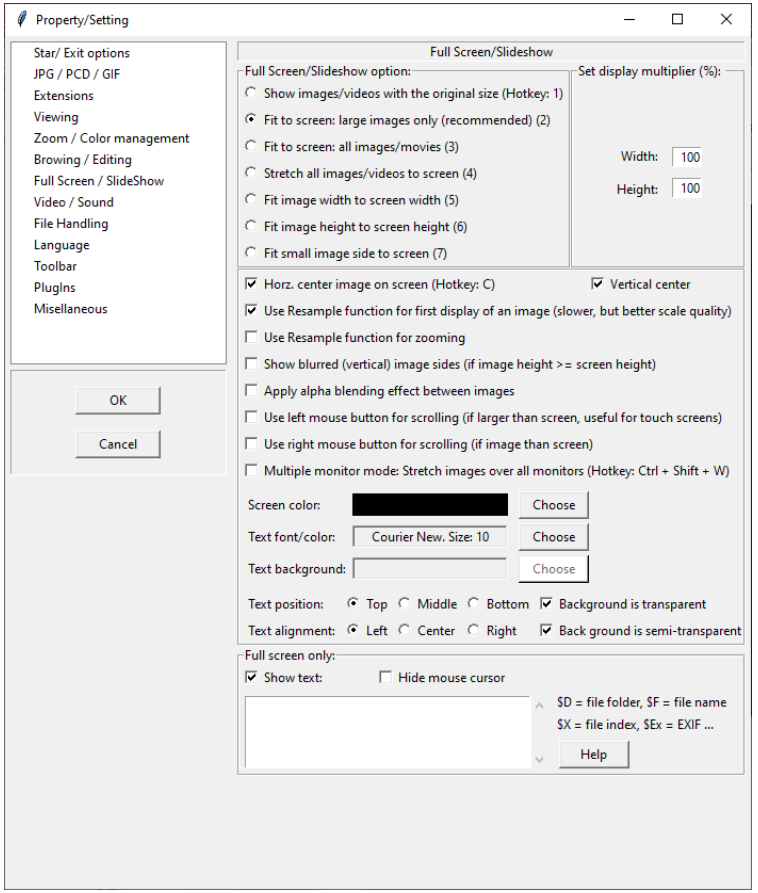
tk_14.py



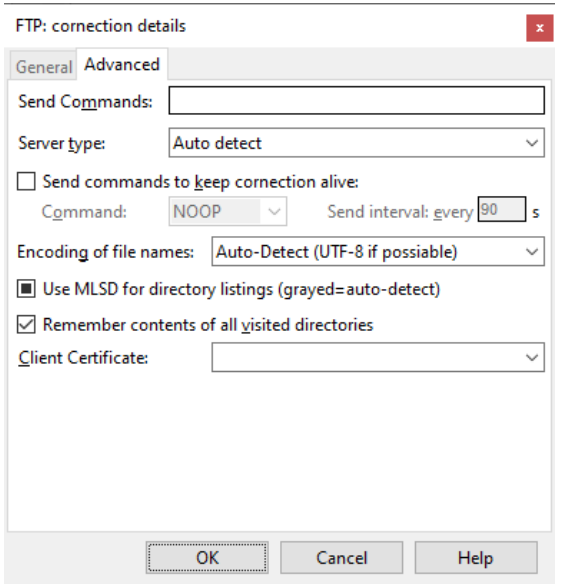
tk_10.py

More Tk Examples

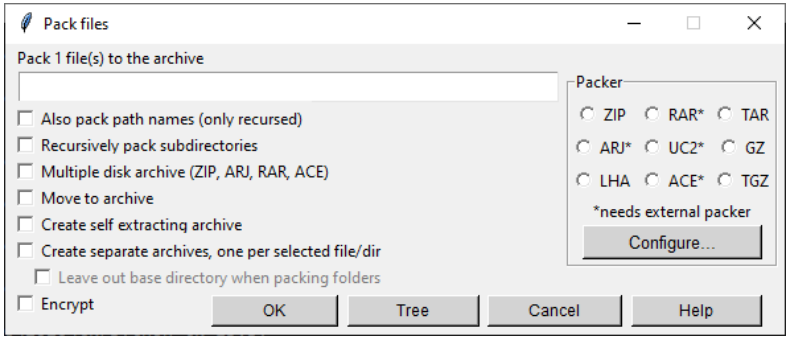
- Examples of Python/Tk implementations of more complex GUIs inspired by real applications



tk_05.py



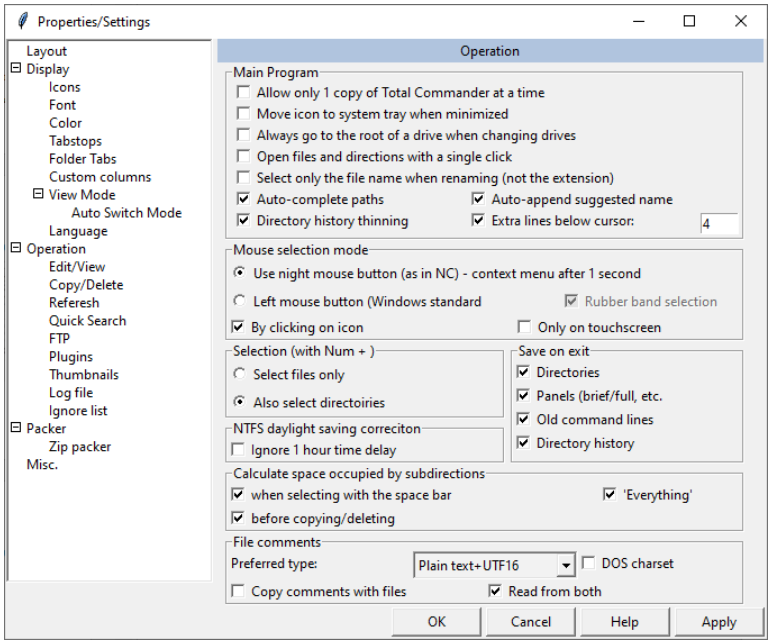
tk_20.py



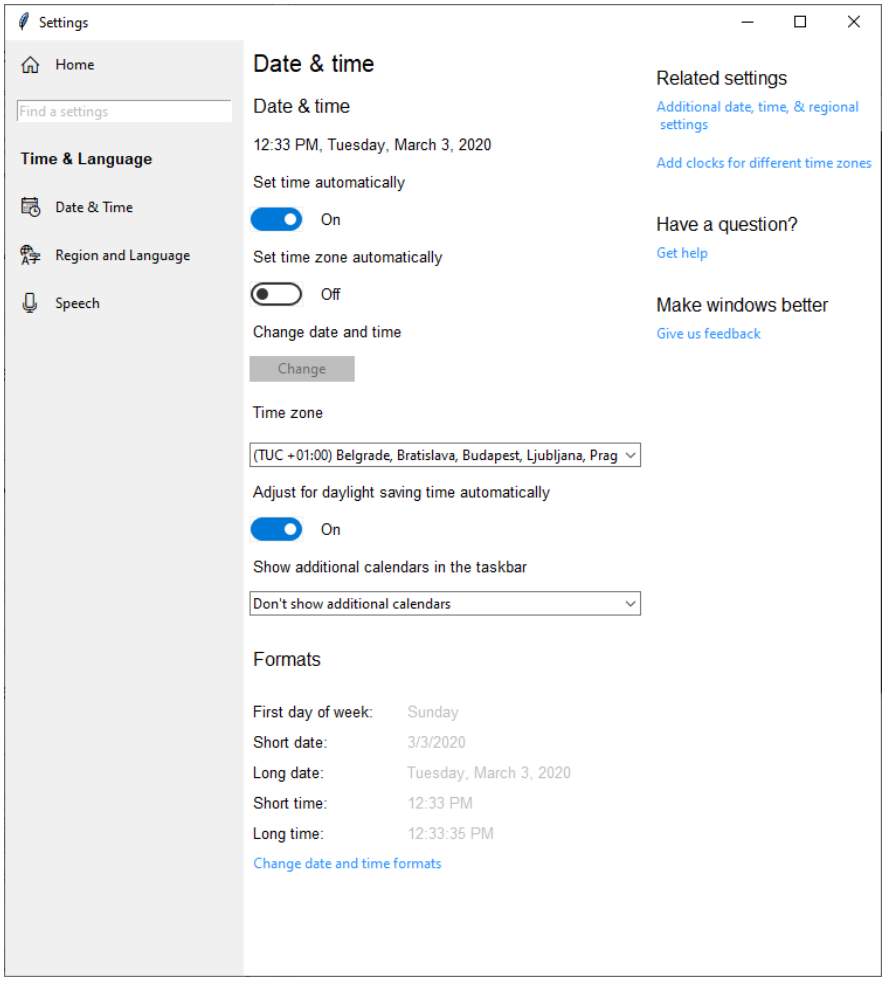
tk_22.py

More Tk Examples

- Examples of Python/Tk implementations of more complex GUIs inspired by real applications



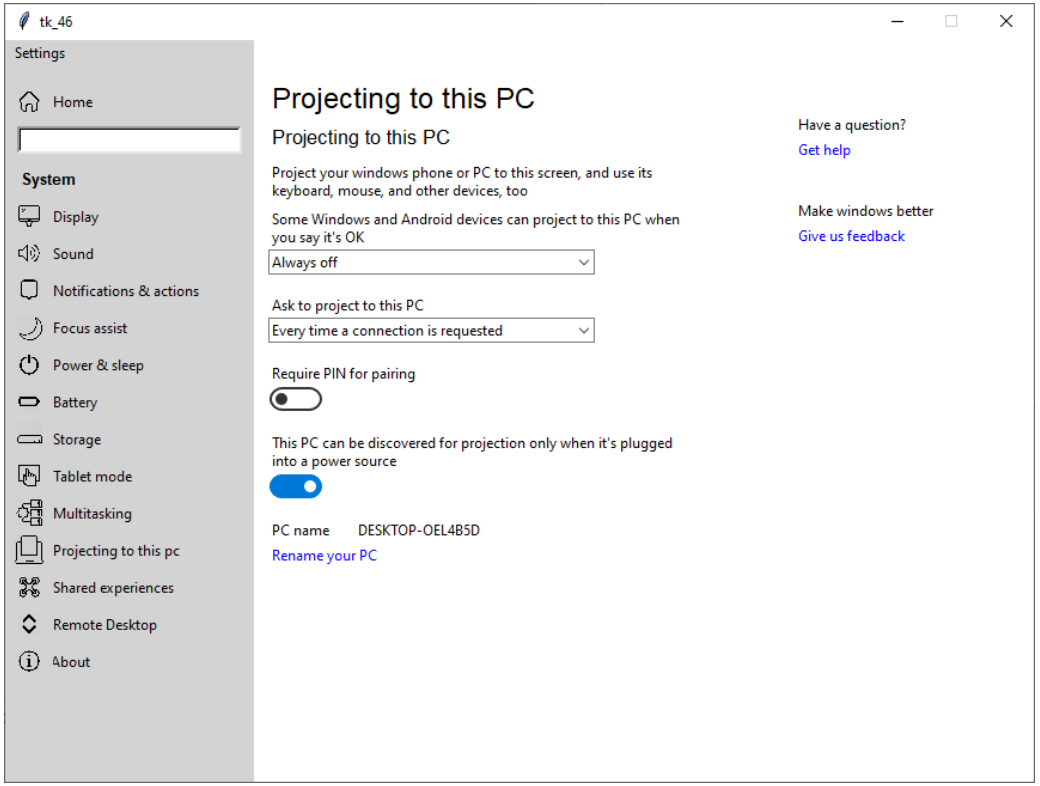
tk_31.py



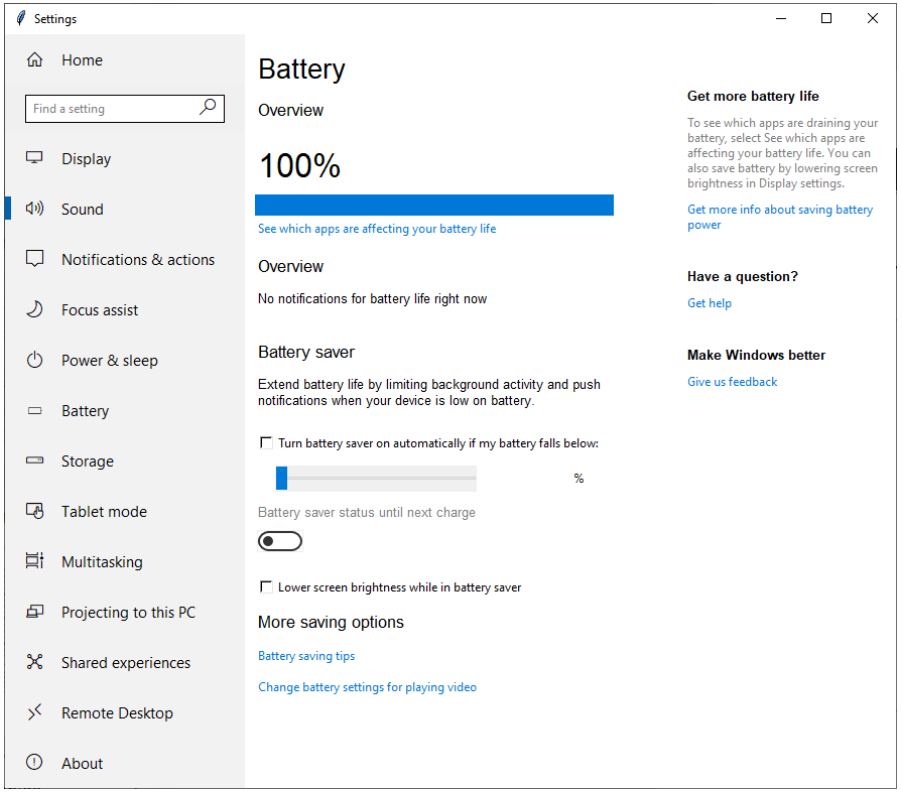
tk_49.py

More Tk Examples

- Examples of Python/Tk implementations of more complex GUIs inspired by real applications



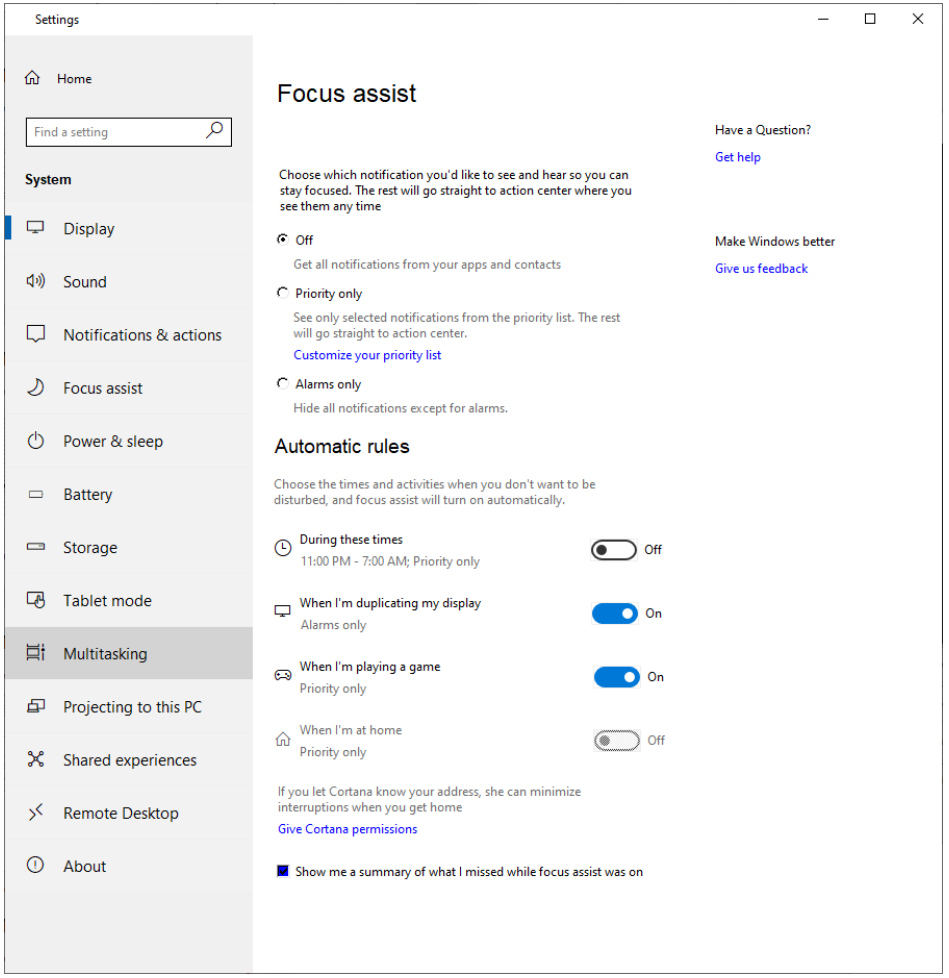
tk_46.py



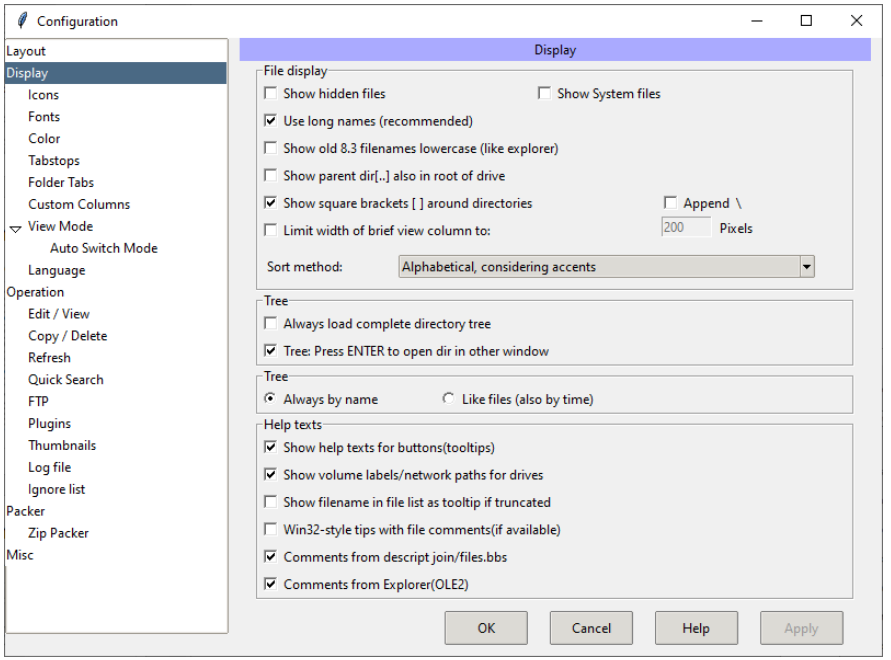
tk_42.py

More Tk Examples

- Examples of Python/Tk implementations of more complex GUIs inspired by real applications



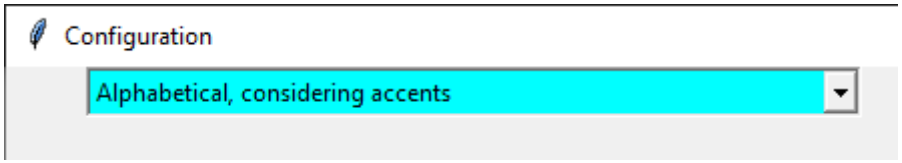
tk_40.py



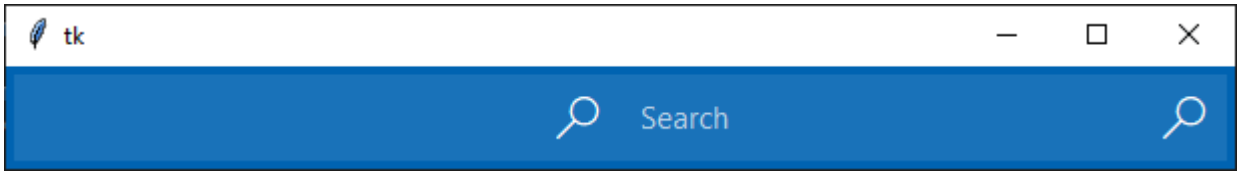
tk_24.py

Hints for Python/Tk Assignments

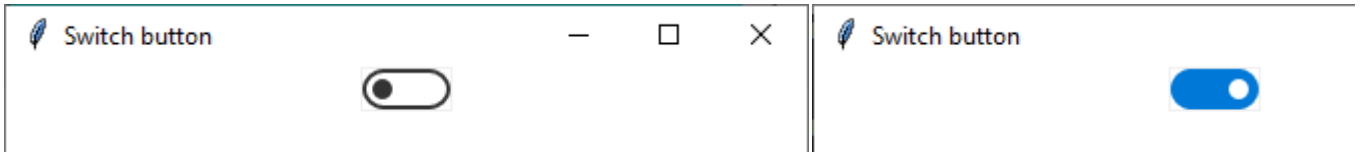
- Source codes with implementations of selected GUI elements in Python/Tk are on the website of this course in the p1_hints.zip file



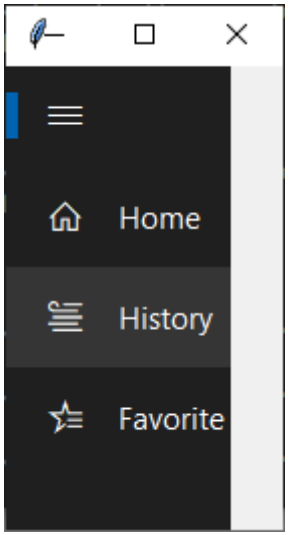
combobox.py



search.py



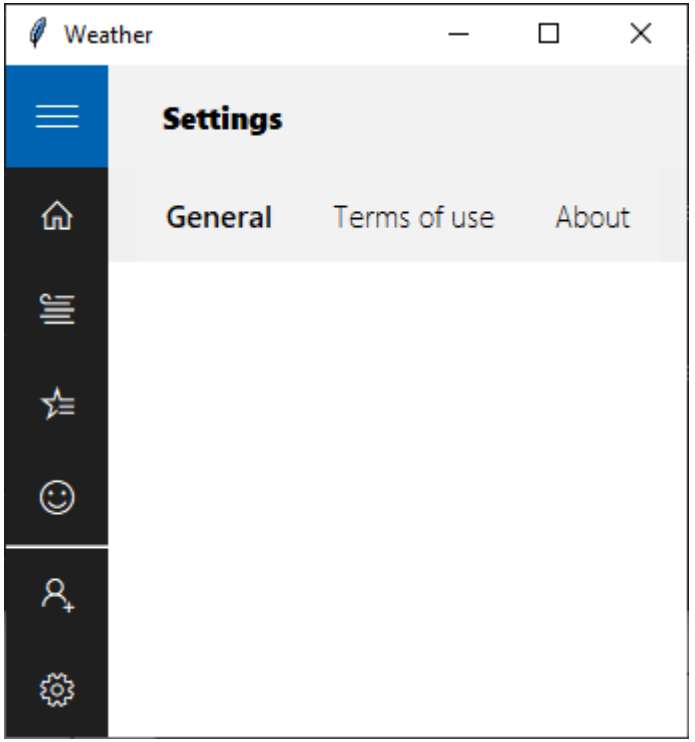
switchbutton.py



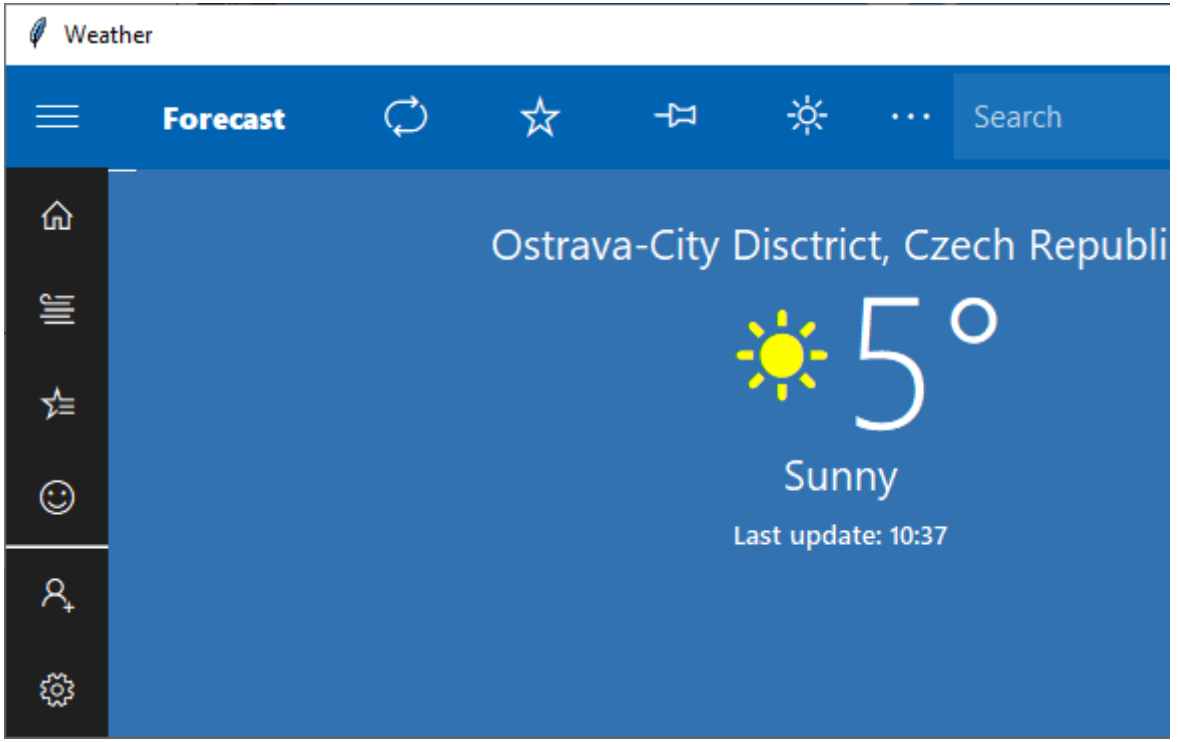
storelikemenu.py

Hints for Python/Tk Assignments

- Source codes with implementations of selected GUI elements in Python/Tk are on the website of this course in the p1_hints.zip file



storelike.py



Tkinter vs. tkinter

The package **Tkinter** has been renamed to **tkinter** in **Python 3**, as well as other modules related to it.

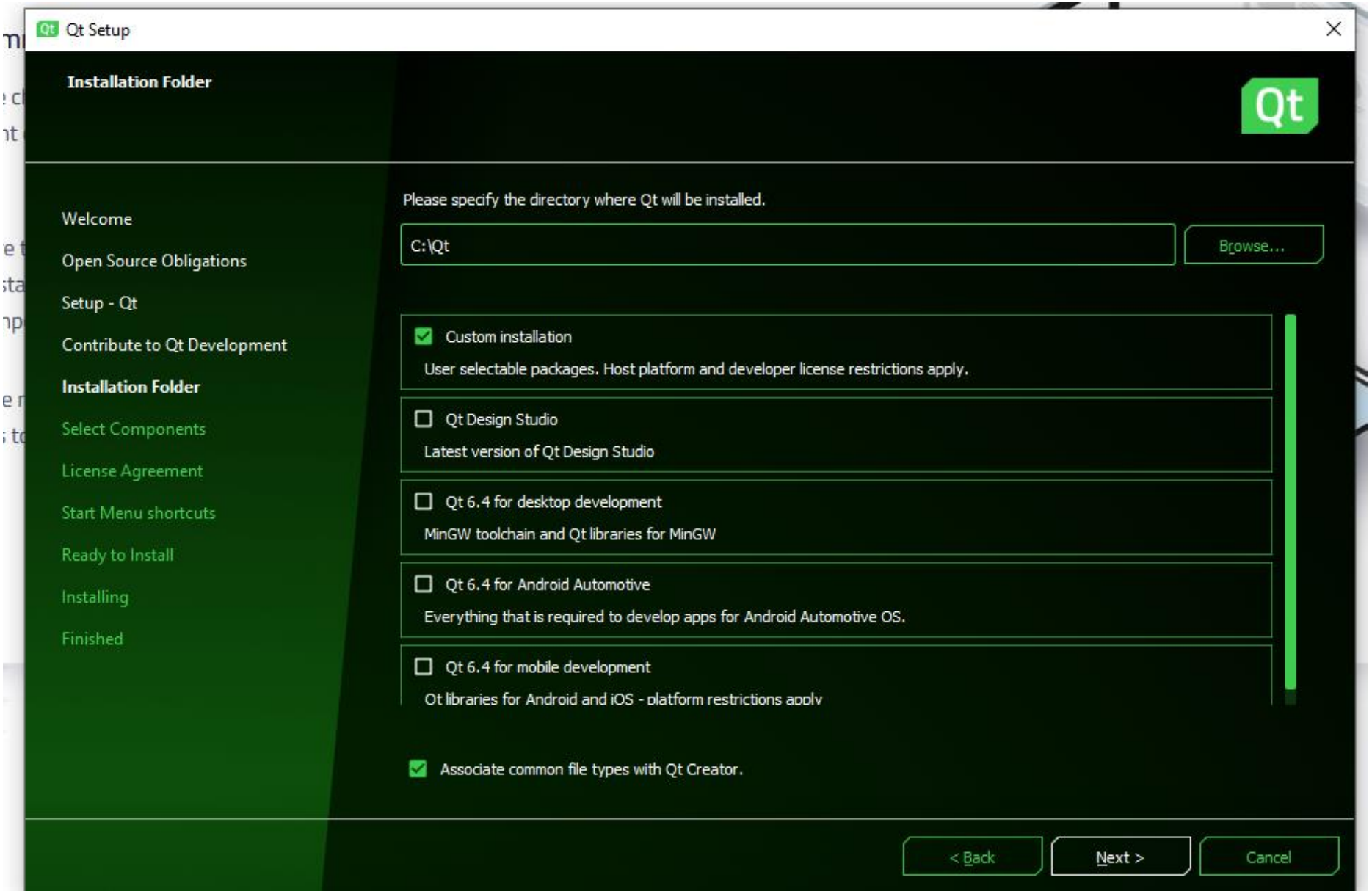
Tkinter → tkinter
tkMessageBox → tkinter.messagebox
tkColorChooser → tkinter.colorchooser
tkFileDialog → tkinter.filedialog
tkCommonDialog → tkinter.commondialog
tkSimpleDialog → tkinter.simpledialog
tkFont → tkinter.font
Tkdnd → tkinter.dnd
ScrolledText → tkinter.scrolledtext
Tix → tkinter.tix
ttk → tkinter.ttk

Qt

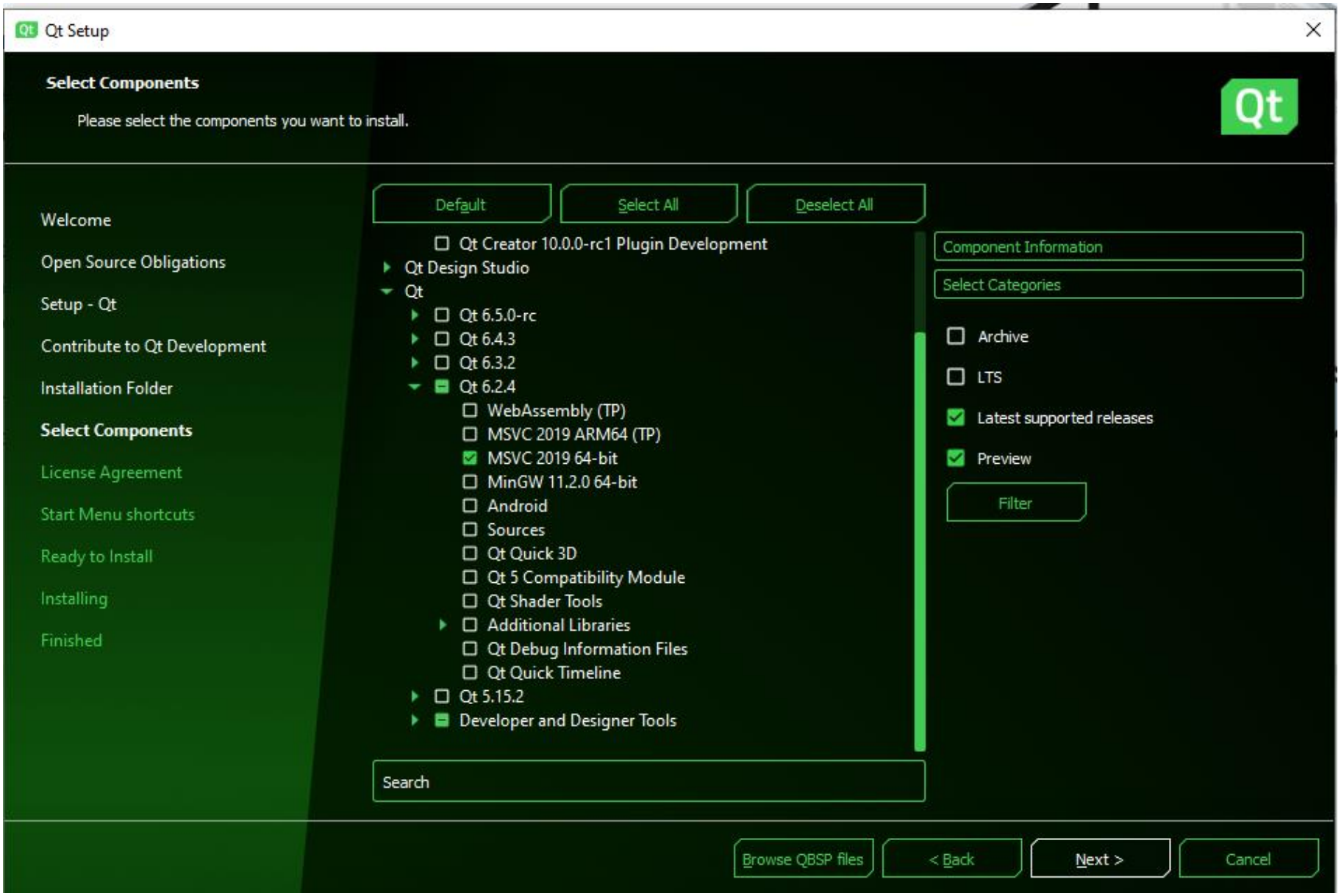
- Before we can start with Qt, we need to install Qt Library and Qt Tools for Visual Studio 2019. This tutorial assumes you have already installed Visual Studio 2019 with C++ development support

- This process consists of two steps
  - step A – Qt library installation procedure
  - step B – Qt Tools for Visual Studio 2019 installation procedure

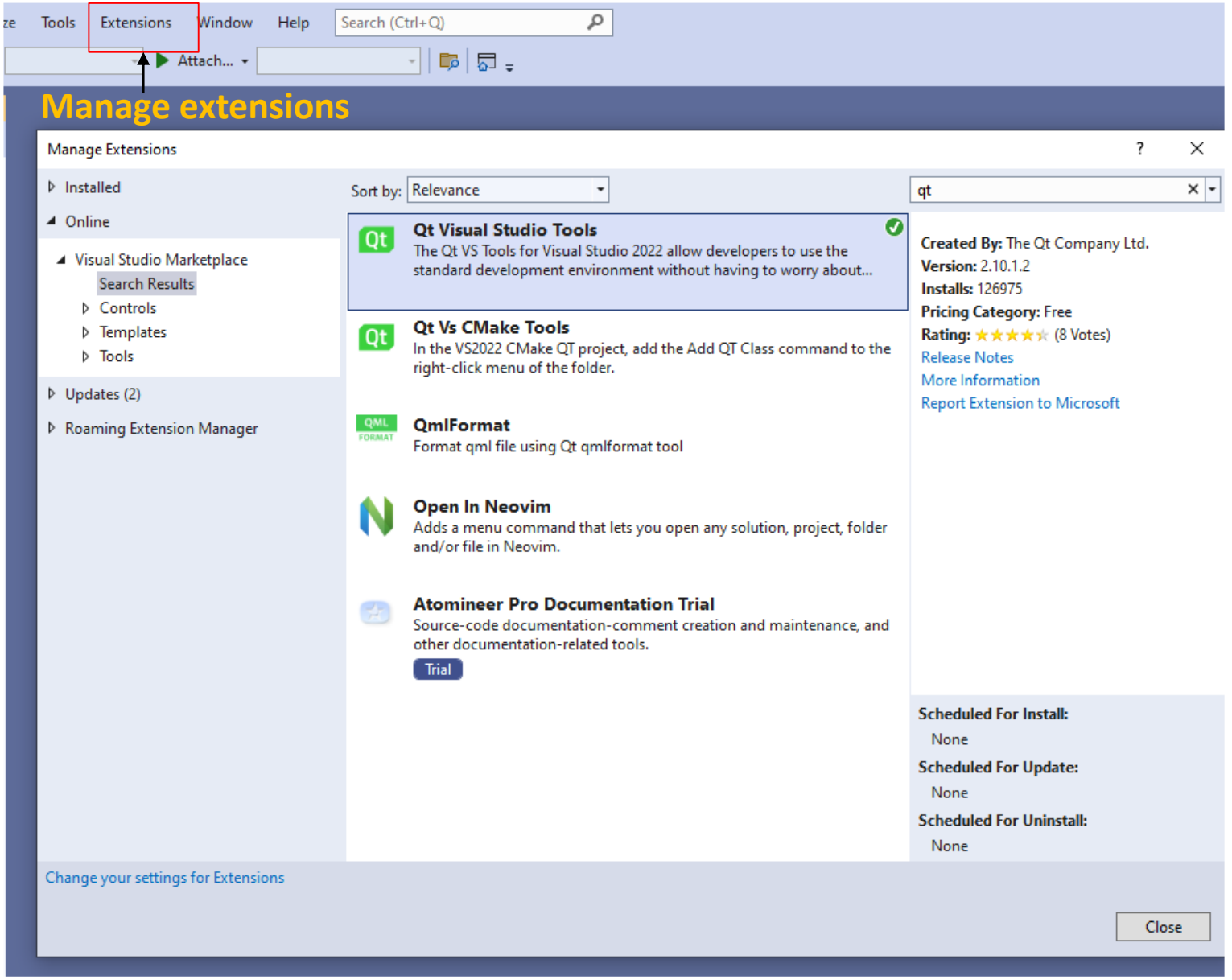
Qt 6.2.4 Installation



Qt 6.2.4 Installation



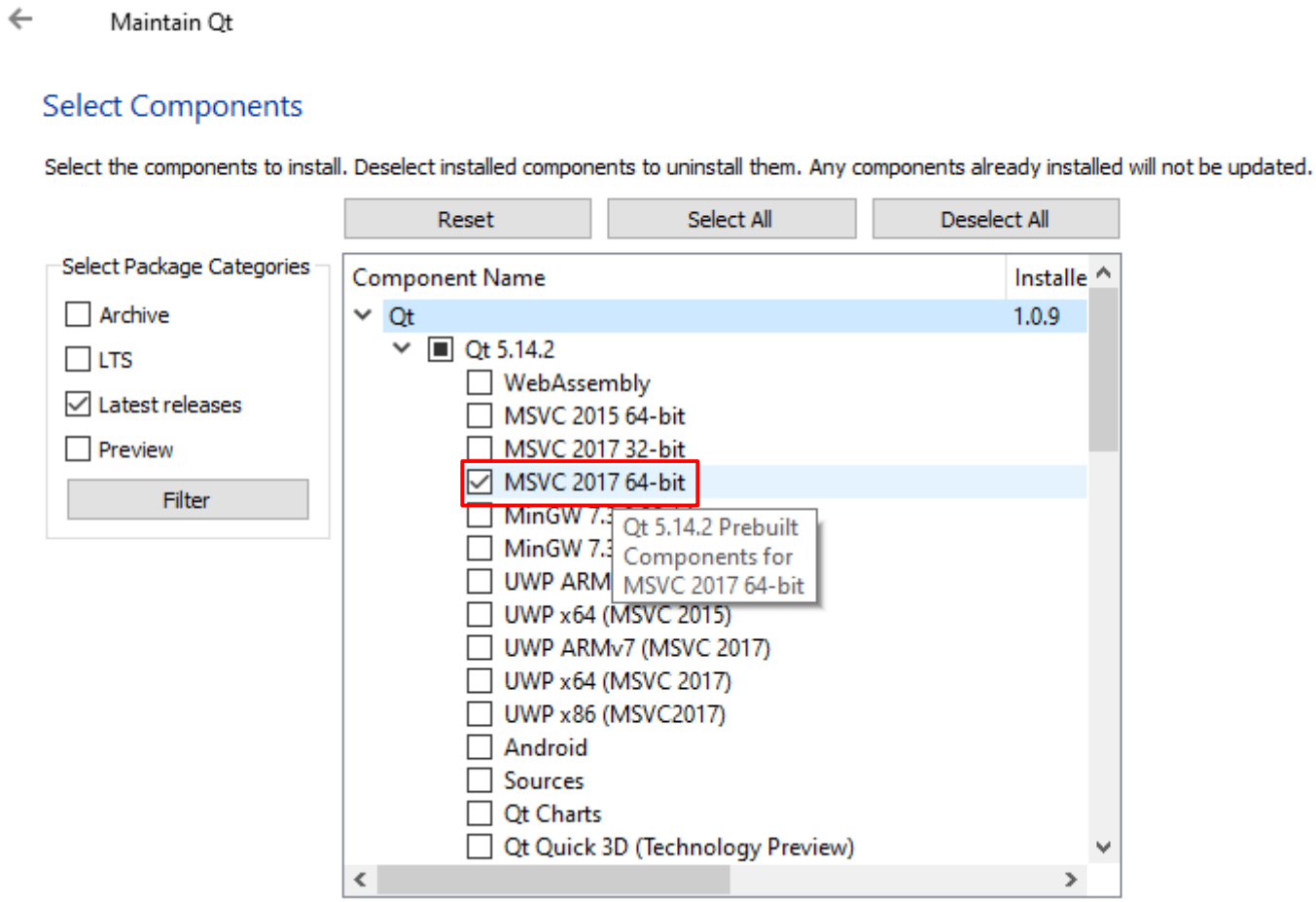
Qt Visual Studio Tools Installation



Qt Library Installation Procedure

- Download qt-unified-windows-x86-3.2.2-online.exe
  https://www.qt.io/download-thank-you?os=windows&hsLang=en
- Run qt-unified-windows-x86-3.2.2-online.exe
- Click next, fill up credentials for a new Qt Account and click Next again.
- Accept the Qt Open Source usage obligations and click Next twice.
- Select Disable sending pseudonymous usage statistic and click Next.
- Accept directory where the Qt will be installed (default is C:\Qt).
- On the Select Components page, select Latest release and under Qt 5.14.2 node, select only the MSVC 2017 64-bit field. All other options should be disabled and click Next (see next slide).
- Accept the License Agreement and click Next.
- Accept the Start Menu shortcut and click Next.
- Click install (installation will use approx. 2 GB of disk space and takes roughly 10 minutes).
- Wait until the installation procedure finishes and proceed with the step B.

Qt Library Installation Procedure



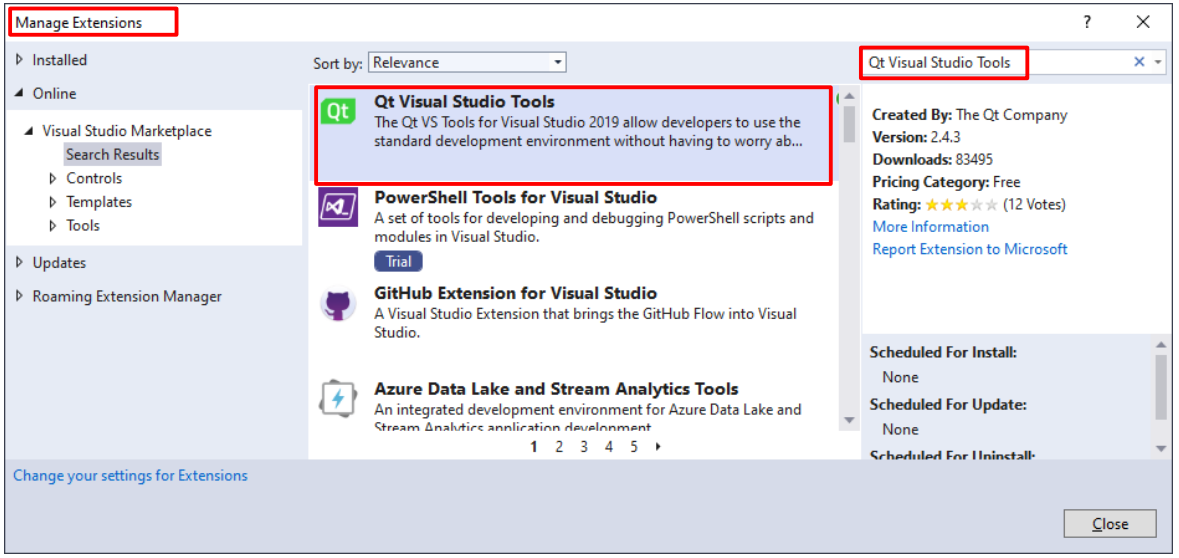
Qt Tools for Visual Studio 2019 Installation Procedure

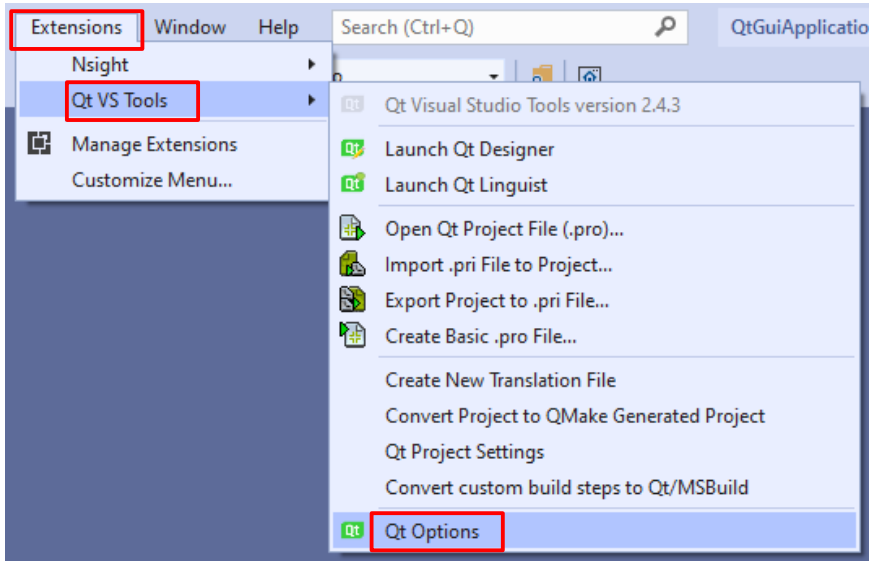
- Run the Visual Studio 2019.
- On the welcome page, click Continue without code.
- From the main menu, select Extensions and Manage Extensions.
- Find Qt Visual Studio Tools (you can use search field) and click Download.
- Close Visual Studio.
- Click Modify in the pop-up window of VSIX Installer.
- Wait until the installation procedure finishes and click Close.
- Start the Visual Studio again.
- On the welcome page, click Continue without code.
- From the main menu, select Extensions, Qt VS Tools, and Qt Options.
- On the Qt Options page click Add and enter the following text

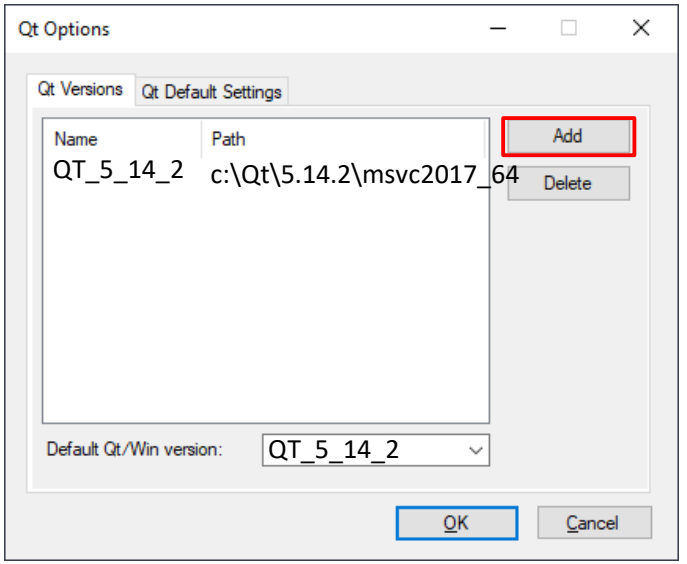
```
Version name: QT_5_14_2
Path: c:\Qt\5.14.2\msvc2017_64
```

and click OK twice.

Qt Tools for Visual Studio 2019 Installation Procedure

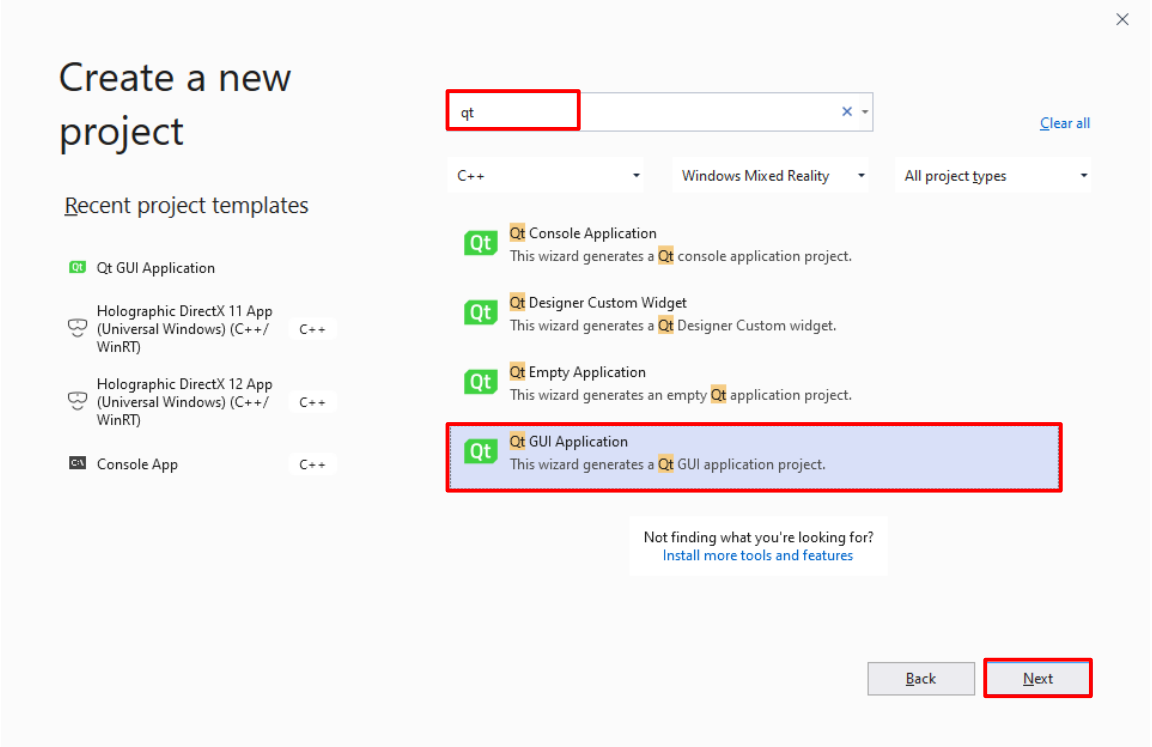






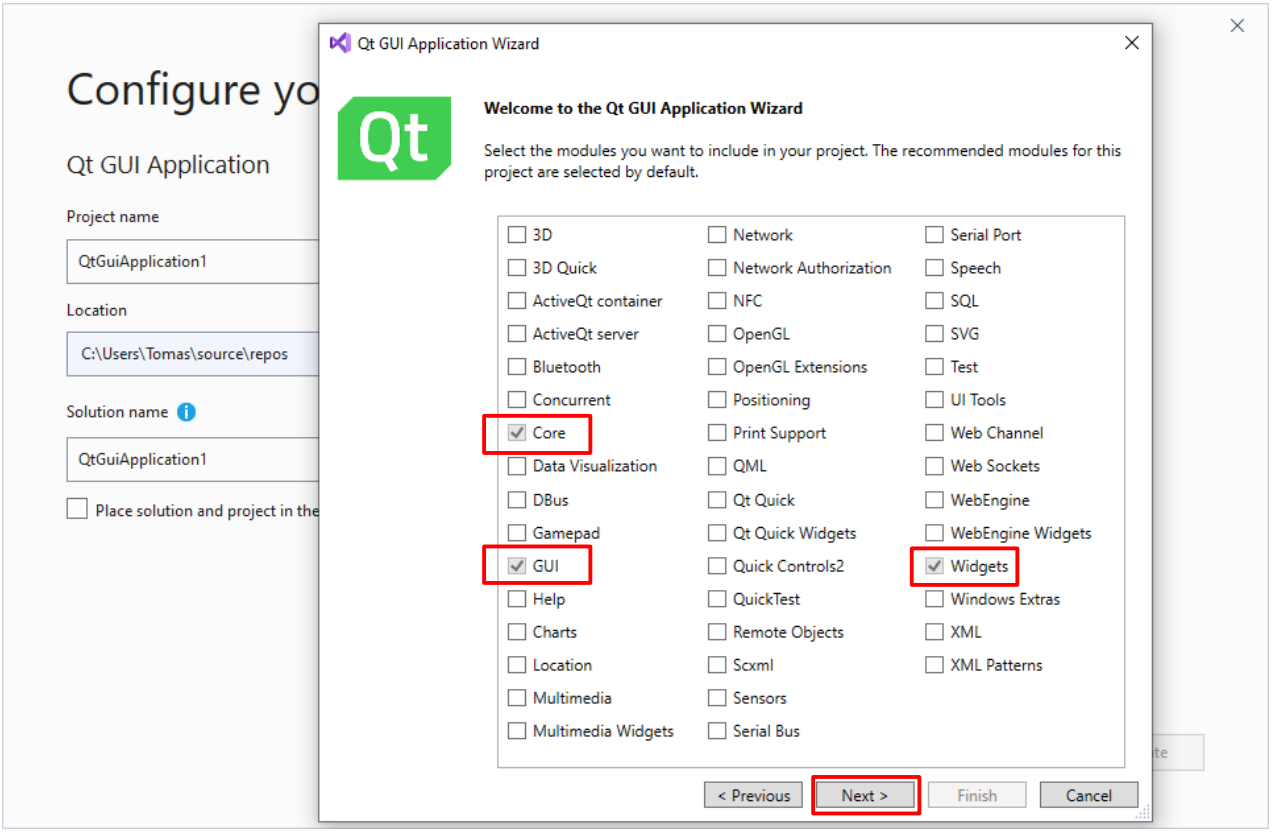
Qt

- Now, you have installed the Qt library and the Qt Tools VS extension
- To begin a new Qt project, start the Visual Studio 2019 and click on Create a new project
- In the search field, type Qt and double click on Qt GUI Application template, which will show up down below and click Next



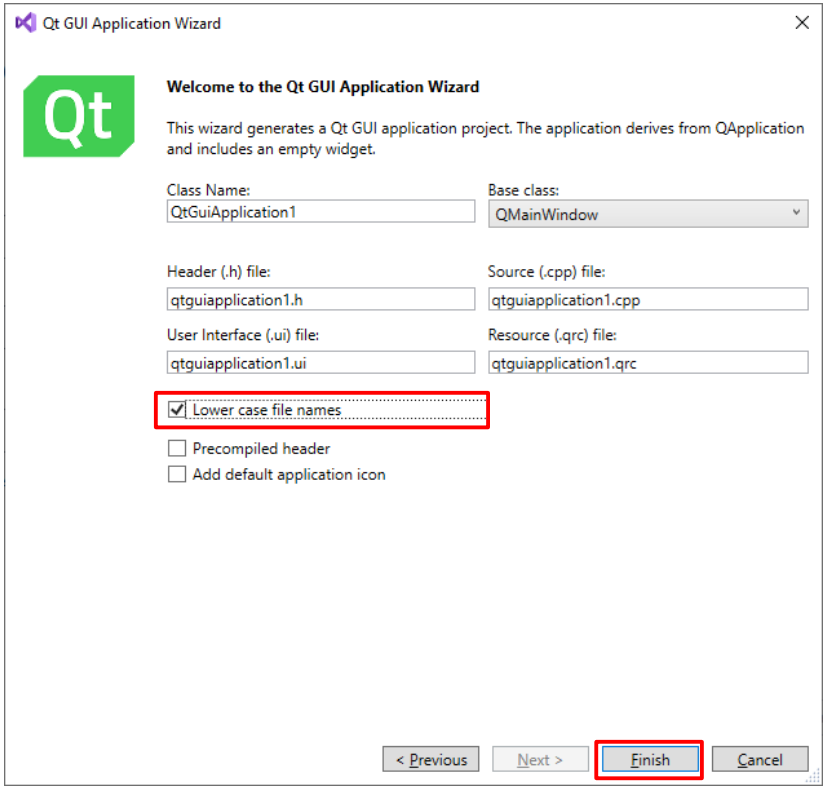
Qt

- In the Qt GUI Application Wizard window, select the following three basic modules (Core, GUI, Widgets) and click Next…



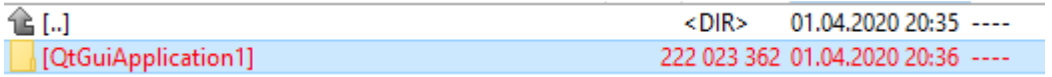
Qt

- Named the main window class as you want and click Finish

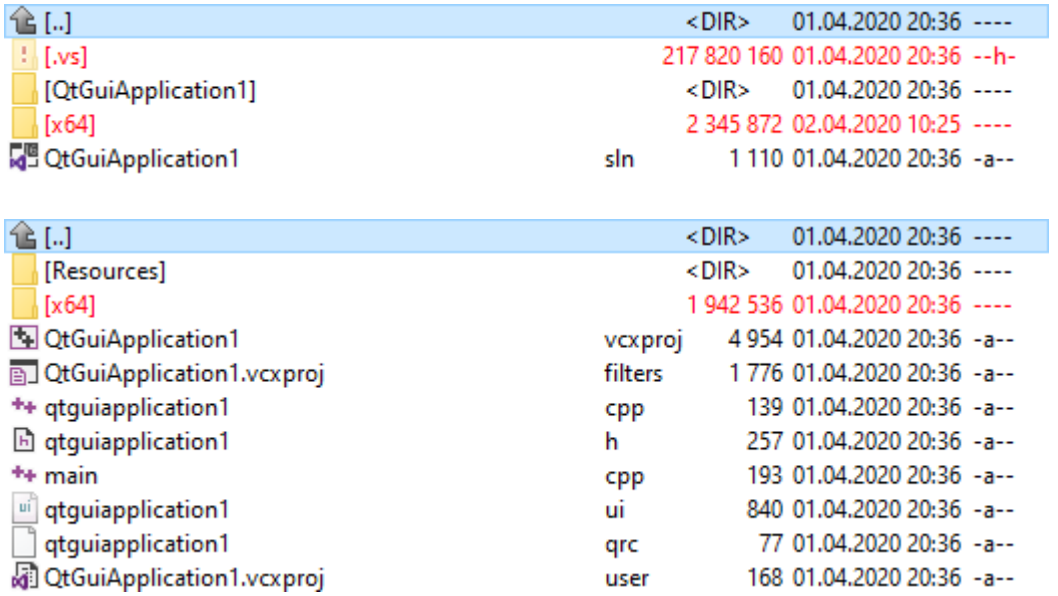


VS Project Cleanup

- Even a newly created project in the Visual Studio may take up several hundred megabytes of space

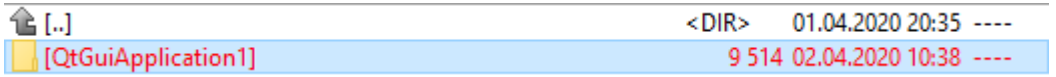


- The majority of these files are unnecessary to rebuild the project. You can delete the following directories (.vs (it is hidden), x64)…

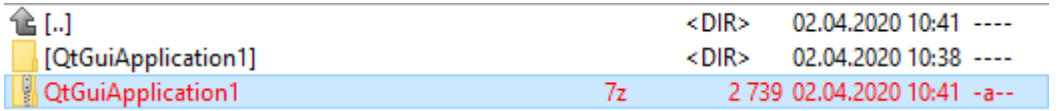


VS Project Cleanup

- Now, the project take up several kilobytes of space



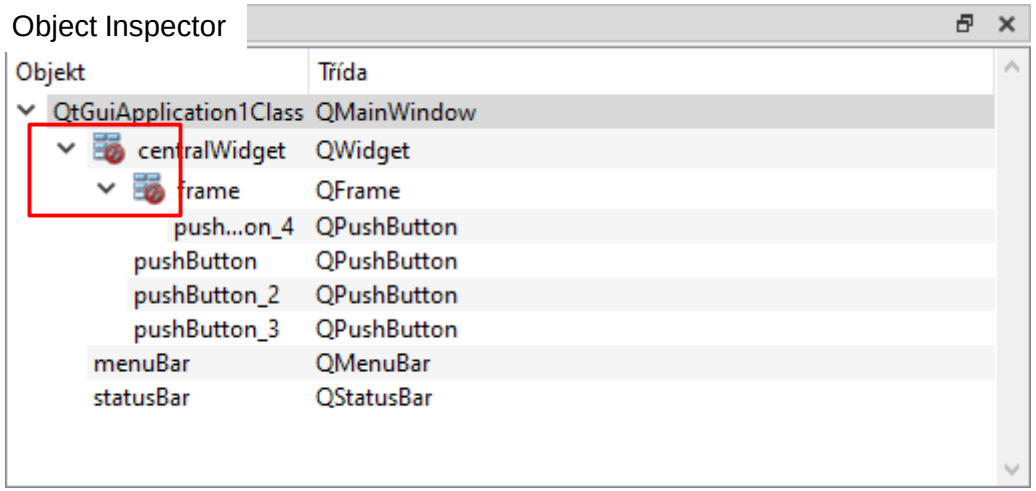
- If you compress the project, you will end with something really small what can be easily transferred

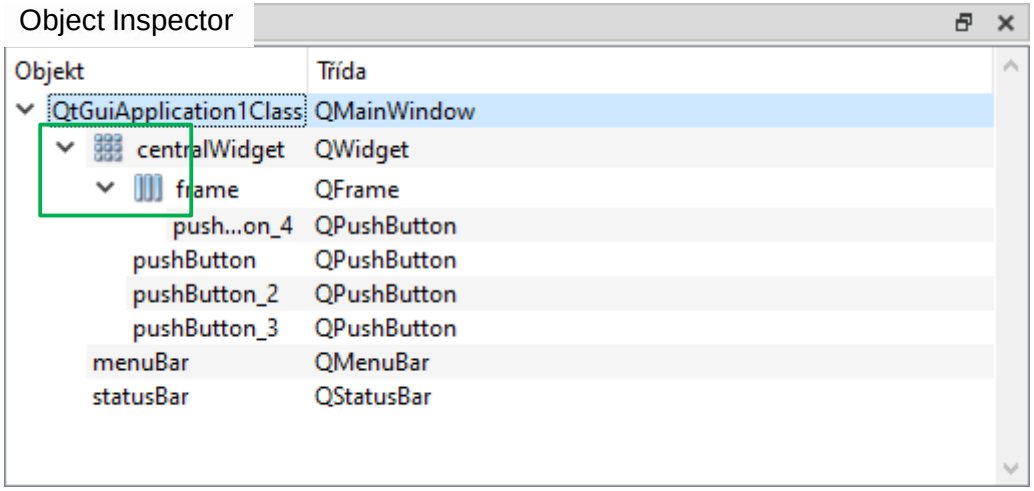


- As a result, we shrunk the project folder from 212 MB to 2.7 KB

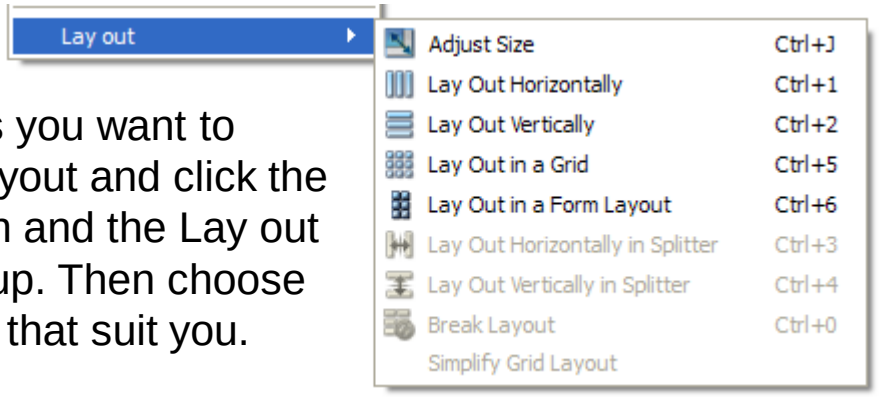
Qt Designer

- In the Qt Designer and also in the C++ code, every container with one or more widgets inside must have a layout assignment





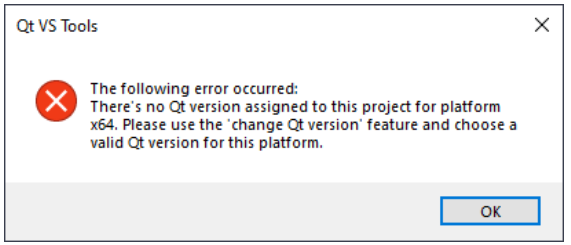
Wrong → Correct



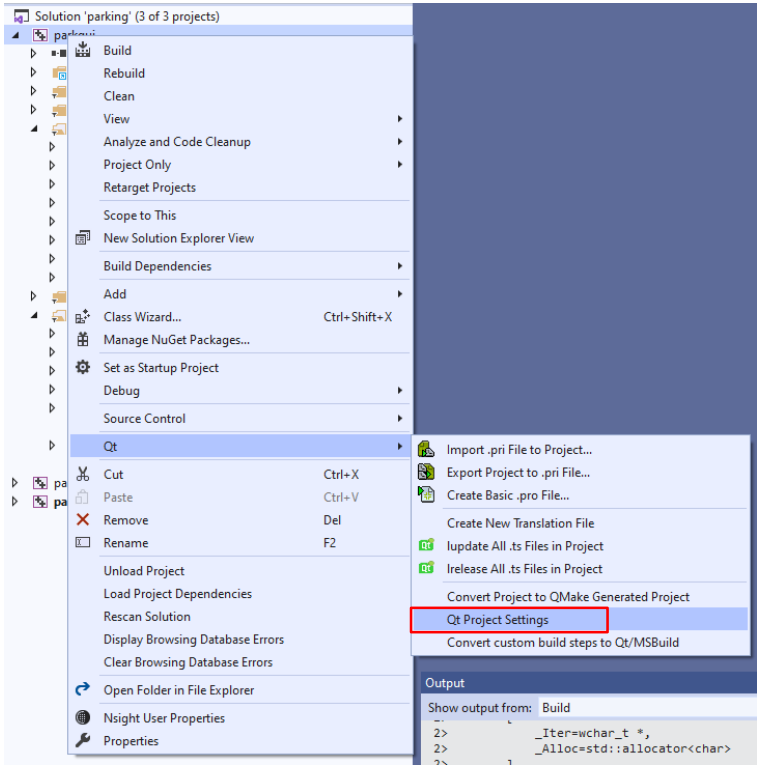
Select the widgets you want to add/change the layout and click the right mouse button and the Lay out submenu will popup. Then choose one of the layouts that suit you.

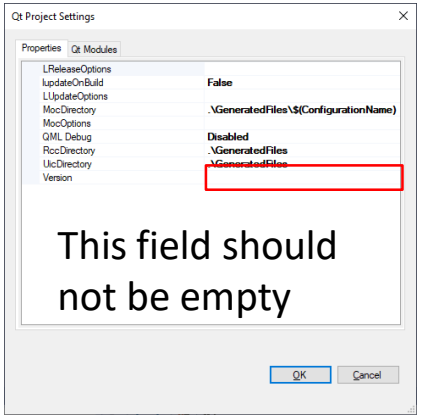
VS Project Fix

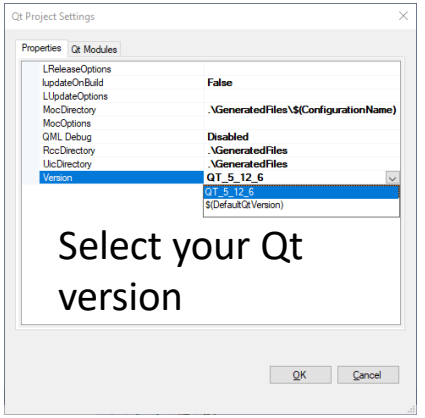
- If you will receive the following error message



- You can fix that error by setting the proper Qt version

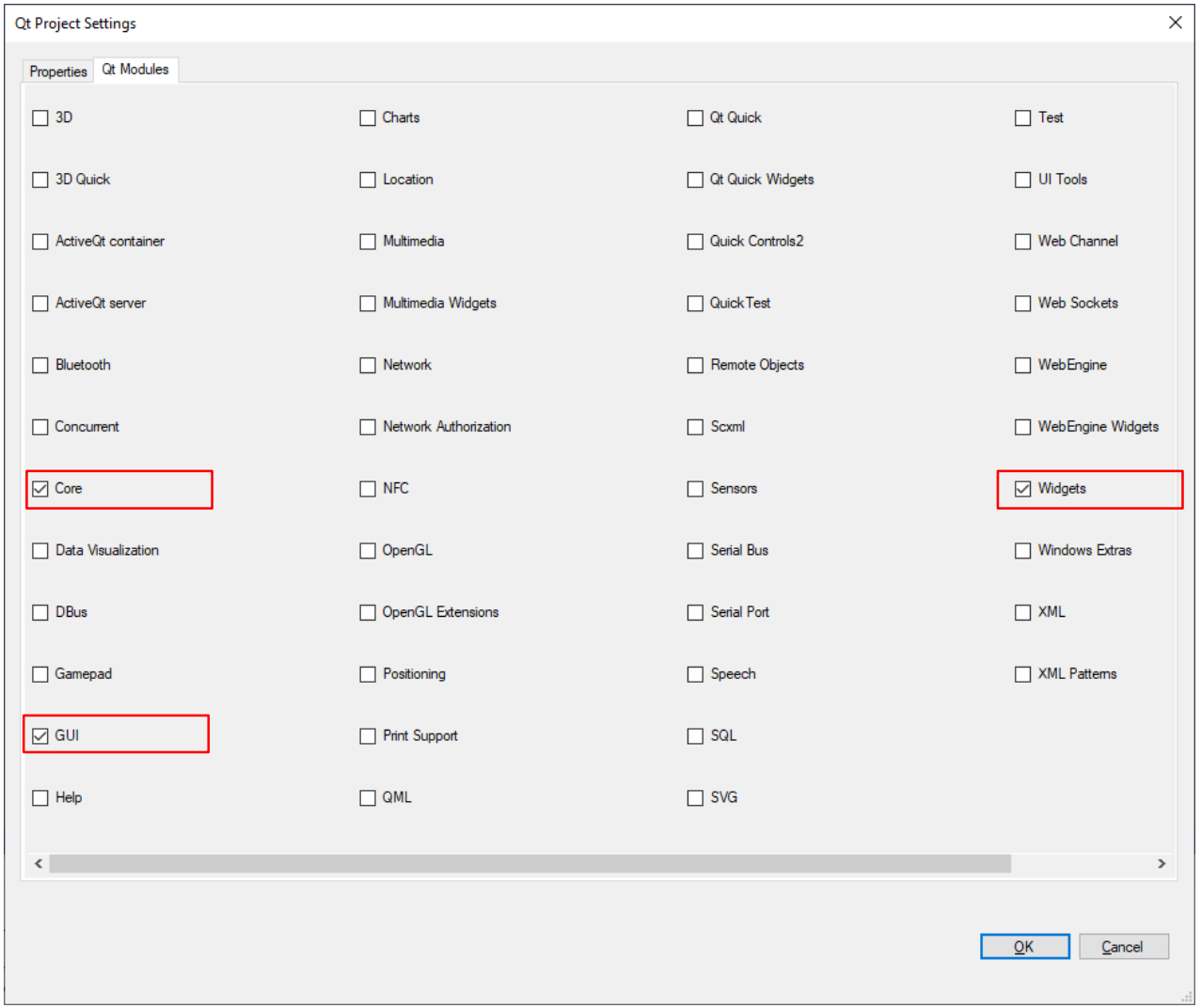




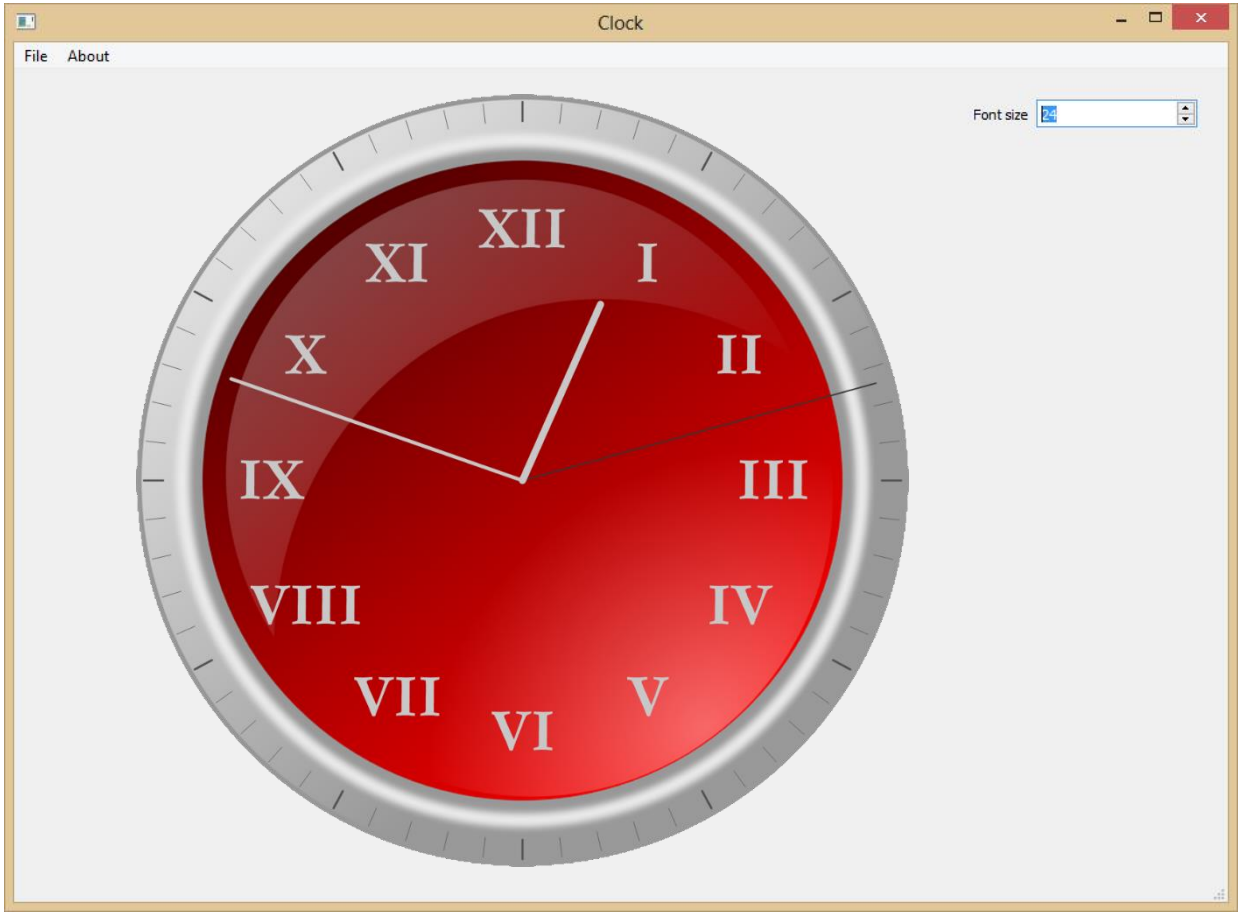


VS Project Fix

- Also check the modules (Core, GUI, and Widgets should be enabled)



Qt Wall Clock



Qt Wall Clock

- We want to establish the connection between the spin box controlling font size and our clock widget:
  - Find an appropriate **signal** coming from the QSpinBox class. We have the following choices:

    (from http://qt-project.org/doc/qt-4.8/qspinbox.html)

void QSpinBox::valueChanged (int *i*) [signal]

This signal is emitted whenever the spin box's value is changed. The new value's integer value is passed in *i*.

void QSpinBox::valueChanged (const QString & *text*) [signal]

This is an overloaded function.

The new value is passed literally in *text* with no prefix() or suffix().

Qt Wall Clock

- We want to establish the connection between the spin box controlling the font size and our clock widget:
  - Create an adequate slot in the clock widget (clockwidget.h):

```
class ClockWidget : public QWidget
{
    Q_OBJECT
    …
    protected slots:
        void font_size_change ( const int i );
    …
};
```

Qt Wall Clock

- We want to establish the connection between the spin box controlling the font size and our clock widget:
  - … and the implementation (clockwidget.cpp):

```
void ClockWidget::font_size_change( const int i )
{
    font_size_ = std::max( i, 1 );
    update();
}
```

  - Don't forget to call **update**() which schedules a paint event for processing when Qt returns to the main event loop.
  - Calling update() several times normally results in just one paintEvent() call.

Qt Wall Clock

- We want to establish the connection between the spin box controlling the font size and our clock widget:
  - … and the implementation (clockwidget.cpp):

```
void ClockWidget::font_size_change( const int i )
{
    font_size_ = std::max( i, 1 );
    update();
}
```

  - Don't forget to call **update**() which schedules a paint event for processing when Qt returns to the main event loop.
  - Calling update() several times normally results in just one paintEvent() call.

Qt Wall Clock

- We want to establish the connection between the spin box controlling the font size and our clock widget:
  - … finally, we connect both widgets (testqt.cpp):

```
testqt::testqt( QWidget * parent ): QMainWindow( parent )
{
…
connect( ui.sbFontSize, SIGNAL( valueChanged( int ) ), clock,
 SLOT( font_size_change( int ) ) );
…
}
```

  - You can check the success of connection in the Output window.

Qt Wall Clock

- In the next step, we want to update the clock according the actual system time:

(clockwidget.h)

```
protected slots:
    void time_change();
private:
    QTimer * timer_;
```

(clockwidget.cpp)

```
timer_ = new QTimer( this );
connect( timer_, SIGNAL( timeout() ), this, SLOT( time_change() ) );
timer_->start( 250 );
```

Qt Wall Clock

- In the next step, we want to update the clock according the actual system time:

(clockwidget.cpp)

```
void ClockWidget::time_change()
{
    QTime current_time = QDateTime::currentDateTime().time();

    second = ( int )( current_time.second() + current_time.msec() * 1e-3 + 0.5    );
    minute = current_time.minute();
    hour = current_time.hour();

    update();
}
```

Qt Wall Clock

- Differences between **update**() and **repaint**():

The method repaint() simply repaints the widget directly by calling paintEvent() immediately, unless updates are disabled or the widget is hidden.

We suggest only using **repaint**() if you need an **immediate repaint**, for example during animation. **In almost all circumstances update() is better**, as it permits Qt to optimize for speed and minimize flicker.

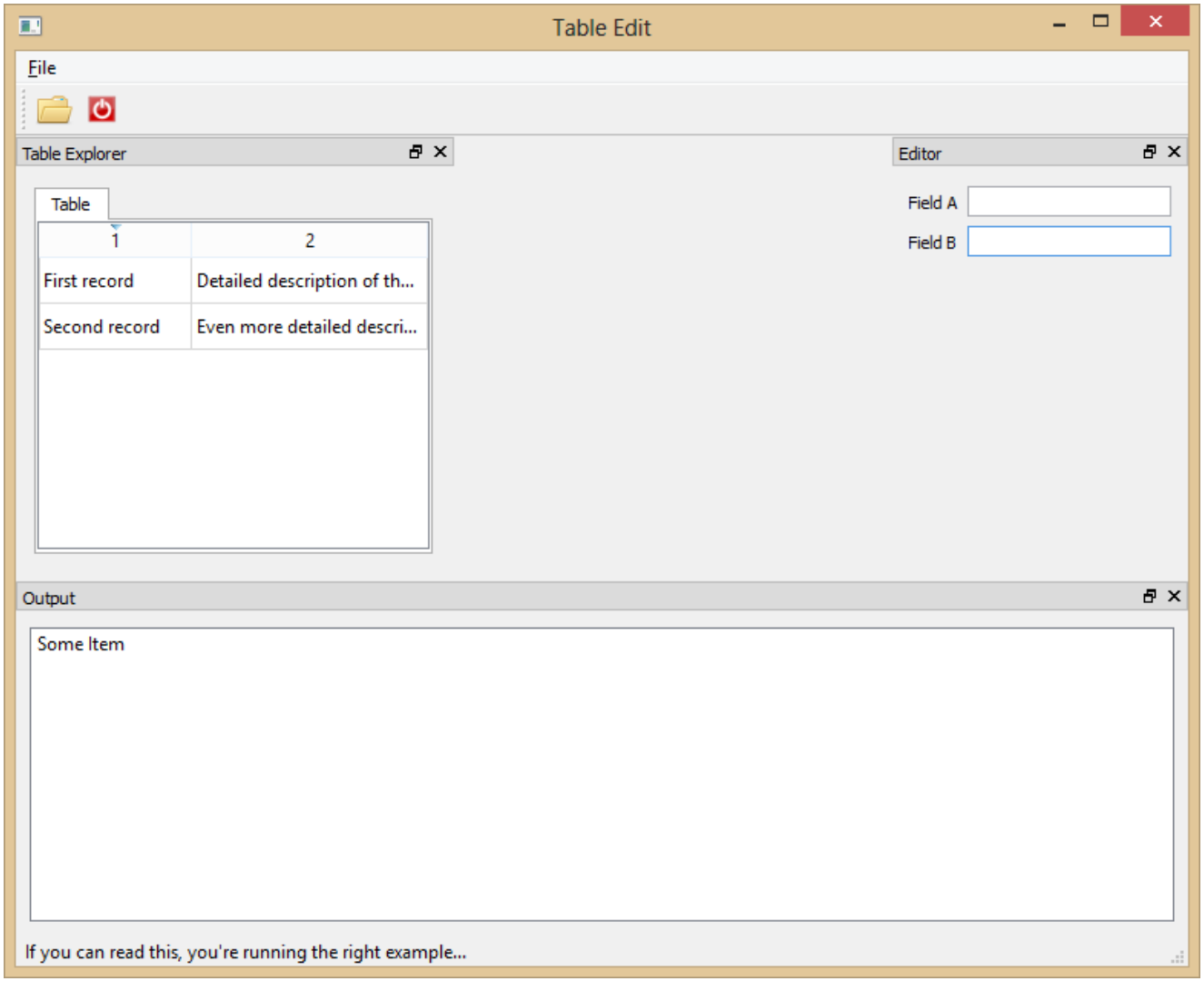
Warning: If you call repaint() in a function which may itself be called from paintEvent(), you may get **infinite recursion**. The update() function never causes recursion.

Qt Splash Screen

- The QSplashScreen widget provides a splash screen that can be shown during application startup

```
int main() {
    QApplication app( argc, argv );
    QPixmap pixmap( ":/myapp/splash.png" );
    QSplashScreen splash( pixmap );
    splash.show();
    app.processEvents();
    Presenter * presenter = new Presenter();
    splash.finish( presenter->InitGUI() );
    return app.exec();
}
```

Qt Table Example



Qt Message Box

- The QMessageBox class provides a modal dialog for informing the user or for asking the user a question and receiving an answer

```
QMessageBox msgBox( this );
msgBox.setWindowTitle( "Exit" );
msgBox.setText( "Do you really want to exit?" );
msgBox.setInformativeText( "All unsaved changes will be lost." );
msgBox.setStandardButtons( QMessageBox::Yes | QMessageBox::No );
msgBox.setDefaultButton( QMessageBox::No );
msgBox.setIcon( QMessageBox::Question );

if ( msgBox.exec() == QMessageBox::Yes )
{
      close();
}
```

Qt Menu Bar

- The QMenuBar class provides a horizontal menu bar. Actions can be added to menus and toolbars, and will automatically keep them in sync.

```
file_menu_ = menuBar()->addMenu( ( "&File" ) );

open_action_ = new QAction( ( "&Open..." ), this );
open_action_->setIcon( QIcon( ":/testqt2/open" ) );
file_menu_->addAction( open_action_ );
ui.mainToolBar->addAction( open_action_ );
connect( open_action_, SIGNAL( triggered() ), this, SLOT( openFile() ) );

file_menu_->addSeparator();

exit_action_ = new QAction( ( "&Exit" ), this );
exit_action_->setIcon( QIcon( ":/testqt2/exit" ) );
exit_action_->setShortcut( QKeySequence( Qt::ALT + Qt::Key_Q ) );
file_menu_->addAction( exit_action_ );
ui.mainToolBar->addAction( exit_action_ );
connect( exit_action_, SIGNAL( triggered() ), this, SLOT( exit() ) );
```

Qt Table Model

QAbstractTableModel provides a standard interface for models that represent their data as a two-dimensional array of items. It is not used directly, but must be subclassed.
QAbstractItemModel - class provides the abstract interface for item model classes.
Other similar class: QAbstractListModel - a model to contain a single column of data.

```
class TableModel : public QAbstractTableModel
{
Q_OBJECT
public:
TableModel( QObject * parent = NULL );

/*
When subclassing QAbstractTableModel, you must implement rowCount(),
columnCount(), and data().
*/
int rowCount( const QModelIndex & parent ) const;
int columnCount( const QModelIndex & parent ) const;
/*
The QVariant class acts like a union for the most common Qt data types.
*/
QVariant data( const QModelIndex & index, int role = Qt::DisplayRole ) const;

private:
QList<QPair<QString, QString>> list;
};
```

Qt Table View

QAbstractTableModel provides a standard interface for models that represent their data as a two-dimensional array of items. It is not used directly, but must be subclassed.
QAbstractItemModel - class provides the abstract interface for item model classes.
Other similar class: QAbstractListModel - a model to contain a single column of data.

```
class TableWidget : public QTabWidget
{
Q_OBJECT
public:
TableWidget( QWidget * parent = NULL );

private:
TableModel * table_model_;
};
```

Qt Table View

QAbstractTableModel provides a standard interface for models that represent their data as a two-dimensional array of items. It is not used directly, but must be subclassed.
QAbstractItemModel - class provides the abstract interface for item model classes.
Other similar class: QAbstractListModel - a model to contain a single column of data.

```
TableWidget::TableWidget( QWidget * parent ) : QTabWidget( parent )
{
table_model_ = new TableModel( this );

QTableView * table_view = new QTableView();
table_view->setModel( table_model_ );
table_view->setSortingEnabled( true );
table_view->setSelectionBehavior( QAbstractItemView::SelectRows );
table_view->horizontalHeader()->setStretchLastSection( true );
table_view->verticalHeader()->hide();
table_view->setEditTriggers( QAbstractItemView::NoEditTriggers );
table_view->setSelectionMode( QAbstractItemView::SingleSelection );

addTab( table_view, "Table" );
}
```

Qt Table Model – New Record

To insert a new record we have to add the following methods in the Table Model class:

bool setData(const QModelIndex &index, const QVariant &value, int role = Qt::EditRole);

The setData() function is the function that inserts data into the table, item by item and not row by row.

bool insertRows(int position, int rows, const QModelIndex &index = QModelIndex());

The insertRows() function is called before new data is added, otherwise the data will not be displayed. The beginInsertRows() and endInsertRows() functions are called to ensure all connected views are aware of the changes.

See the example for further reference.

Qt Table Model – Remove Record

To remove a selected record we have to add the following method in the Table Model class:

bool TableModel::removeRows(int position, int rows, const QModelIndex &index)

Don't forget to assign **QSortFilterProxyModel** class providing support for sorting and filtering data.

```
TableWidget::TableWidget( QWidget * parent ) : QTabWidget( parent )
{
…
QSortFilterProxyModel * proxy_model = new QSortFilterProxyModel( this );
proxy_model->setSourceModel( table_model_ );
…
}
```

See the example for further reference.

Qt Threading Basics

Eeach program has one thread when it is started. This thread is called the GUI thread in Qt applications. The Qt GUI must run in this thread. All widgets and several related classes are created and work in GUI thread.

To ensure the GUI interactivity, a secondary thread is commonly used to offload processing work from the main thread.

There are basically two use cases for threads:

- Make processing faster by making use of multicore processors.
- Keep the GUI thread or other time critical threads responsive by offloading long lasting processing or blocking calls to other threads.

Qt Threading Basics

It is easy to start other threads, but very hard to ensure that all shared data remains consistent. Before creating threads to solve certain problems, possible alternatives should be considered.

QEventLoop::processEvents() - Calling this method repeatedly during a time-consuming calculation prevents GUI blocking. However, this solution doesn't scale well because the call to processEvents() may occur too often, or not often enough, depending on hardware.
QTimer - Background processing can sometimes be done conveniently using a timer to schedule execution of a slot at some point in the future. A timer with an interval of 0 will time out as soon as there are no more events to process.
QSocketNotifier, QNetworkAccessManager, QIODevice::readyRead() - This is an alternative to having one or multiple threads, each with a blocking read on a slow network connection. As long as the calculation in response to a chunk of network data can be executed quickly, this reactive design is better than synchronous waiting in threads. Reactive design is less error prone and energy efficient than threading. In many cases there are also performance benefits.

Qt Threading Basics

Lifetime of thread	Development task	Solution
One call	Run one method within another thread and quit the thread when the method is finished.	• Write a function and run it with QtConcurrent::run() • Derive a class from QRunnable and run it in the global thread pool with QThreadPool::globalInstance()->start() • Derive a class from QThread, reimplement the QThread::run() method and use QThread::start() to run it.
One call	Operations are to be performed on all items of a container.	QtConcurrent provides the map() function for applying operations on every container element, filter() for selecting container elements, and the option of specifying a reduce function for combining the remaining elements.
One call	A long running operation has to be put in another thread. During the course of processing, status information should be sent to the GUI thread.	Use QThread, reimplement run and emit signals as needed. Connect the signals to the GUI thread's slots using queued signal/slot connections.
Permanent	Have an object living in another thread and let it perform different tasks upon request. This means communication to and from the worker thread is required.	Derive a class from QObject and implement the necessary slots and signals, move the object to a thread with a running event loop and communicate with the object over queued signal/slot connections.
Permanent	Have an object living in another thread, let the object perform repeated tasks such as polling a port and enable communication with the GUI thread.	Same as above but also use a timer in the worker thread to implement polling. However, the best solution for polling is to avoid it completely. Sometimes using QSocketNotifier is an alternative.

QThread

Method **start**(priority) begins execution of the thread by calling **run**(). The operating system will schedule the thread according to the priority parameter.

QThread will notify you via a signal when the thread is started() and finished(), or you can use isFinished() and isRunning() to query the state of the thread.

The static functions currentThreadId() and currentThread() return identifiers for the currently executing thread.

Use wait() to block the calling thread, until the other thread has finished execution (or until a specified time has passed).

QMutex

The QMutex class provides access serialization between threads.

The purpose of a QMutex is to protect an object, data structure or section of code so that only one thread can access it at a time.

When you call lock() in a thread, other threads that try to call lock() in the same place will block until the thread that got the lock calls unlock(). A non-blocking alternative to lock() is tryLock().

It is usually best to use a mutex with a **QMutexLocker** since this makes it easy to ensure that locking and unlocking are performed consistently.

QWaitCondition

The QWaitCondition class provides a condition variable for synchronizing threads.

QWaitCondition allows a thread to tell other threads that some sort of condition has been met. One or many threads can block **waiting** for a QWaitCondition to set a condition with **wakeOne**() or **wakeAll**(). Use wakeOne() to wake one randomly selected thread or wakeAll() to wake them all.

QWaitCondition

bool QWaitCondition::**wait**(Qmutex * lockedMutex, ulong time)
Releases the lockedMutex and waits on the wait condition. The lockedMutex must be initially locked by the calling thread. If lockedMutex is not in a locked state, this function returns immediately. If lockedMutex is a recursive mutex, this function returns immediately. The lockedMutex will be unlocked, and the calling thread will block until either of these conditions is met:

- Another thread signals it using wakeOne() or wakeAll(). This function will return true in this case.
- time milliseconds has elapsed. If time is ULONG_MAX (the default), then the wait will never timeout (the event must be signalled). This function will return false if the wait timed out.

The lockedMutex will be returned to the same locked state. This function is provided to allow the atomic transition from the locked state to the wait state.

QWaitCondition

void QWaitCondition::**wakeAll**()
Wakes all threads waiting on the wait condition. The order in which the threads are woken up depends on the operating system's scheduling policies and cannot be controlled or predicted.

void QWaitCondition::**wakeOne**()
Wakes one thread waiting on the wait condition. The thread that is woken up depends on the operating system's scheduling policies, and cannot be controlled or predicted.

If you want to wake up a specific thread, the solution is typically to use different wait conditions and have different threads wait on different conditions.

Application Design

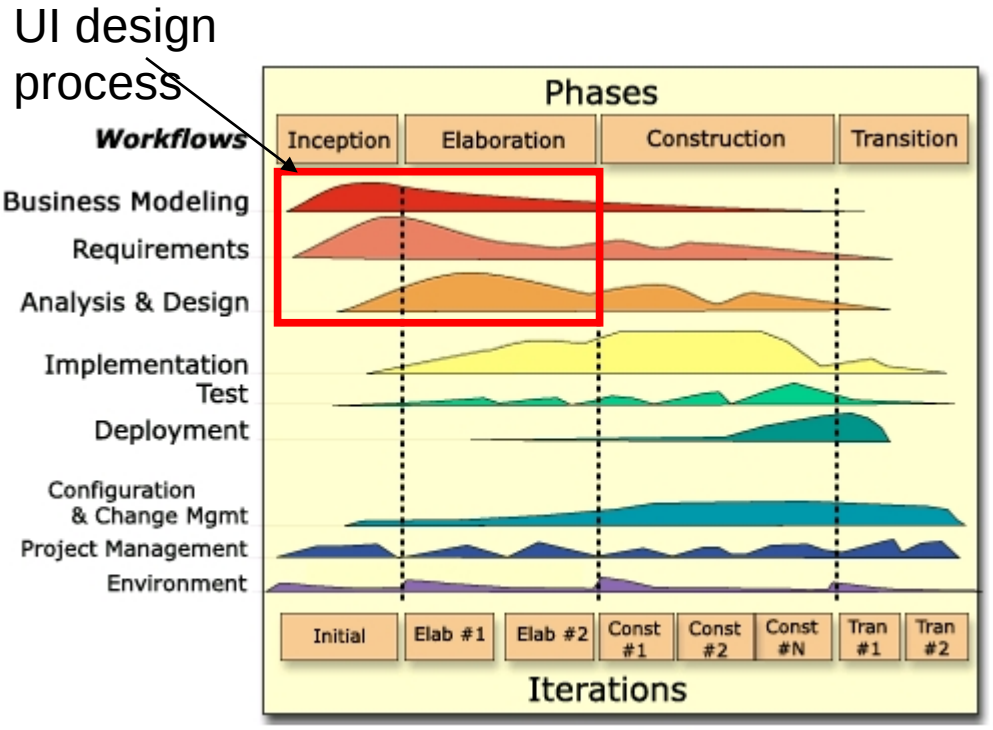
- What should influence the application design:
  - User skills, workflow, habits, and expectations
  - User should be involved in the design process
  - Application execution *flavor* (or *posture*)
    - Sovereign, Transient, Parasitic, Daemonic, and Kiosk
  - Implementation should not dictate the UI design
- Based on RUP
  - Executed in a sequence, iterative

Application Design

- *Sovereign* apps, like MS Word maintain our attention for a length of time
- *Transitory (transient)* apps, like Audio Control Panel, serve us for a short term need and then we move on
- *Daemonic* apps, like Deamon Tools or Antivirus, are background processes that require no direct user interaction
- *Parasitic* (or *auxiliary*) apps, similar to daemonic apps in providing a limited set of functionality, but are shown persistently
- *Kiosk* apps are designed for an interactive computer terminals and protect them from users misuse (touchscreens, virtual keyboards, remote reporting, security features)

Application Design

- RUP Process Architecture



Application Design

- Different user types may use your application
- Target them by different types of widgets:
  - Novice level - Rich menus
  - Expert level - Toolbars
  - Guru level - Command line
- Target those that will pay the most for the SW
- Frequency of use:
  - continual, frequent, occasional, once
- Tolerance of a learning curve
  - none, a little, expected

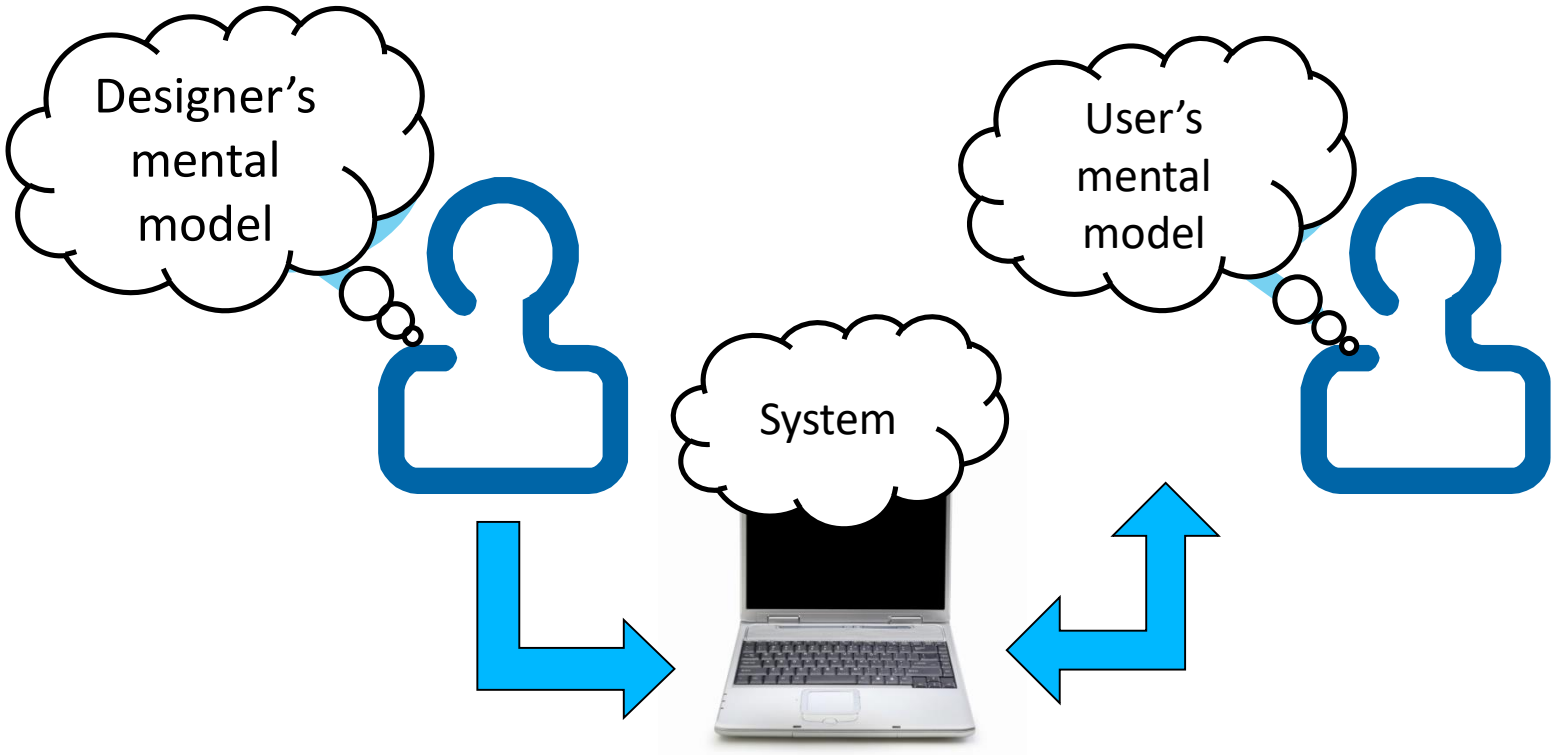
Mental model

- Mental model is the user understanding how the application parts work together
- Mental model is divided into two parts:
  - *Static elements* – previous knowledge, experience, education
  - *Dynamic elements* – based on static elements, users's training and experience with GUIs
- Scottish psychologist Kenneth Craik (1943) - The Nature of Exploration: *The mind constructs "small-scale models" of reality that it uses to reason*, to anticipate events and to underlie explanation.

Mental model

- Philip Johnson-Laird (1989): The reader creates a mental model of the text being read, which simulates the 'world' being described, as the reader understands/interprets it.
- The passages of text that unambiguously produce a single mental model are easier to comprehend; ambiguous passages of text can lead to more than one competing mental model, which can also be deliberately used...

Mental model



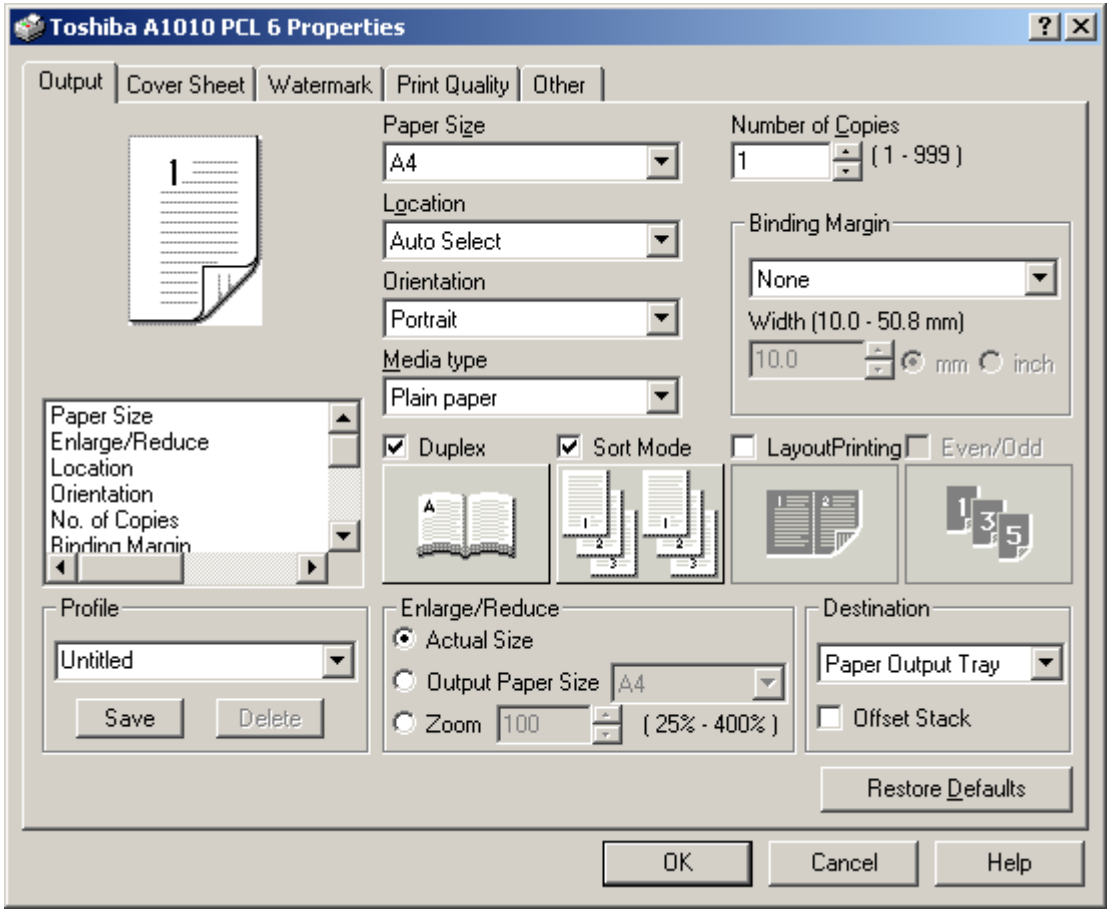
- Worst-case scenario: Developers often have a flawed mental model of their own software and a real user's mental model is quite different
- Expression of mental models: Flow diagrams – a way to express a dynamic systems

Mental model

- When the user discovers the mental model of an application:
  - Sense of confidence
  - Forecasting the behavior in new situations

- In the opposite case the will experience frustration, dubiousness, etc.

Mental model



Gestalt theory

- German: Gestalt – essence or shape of an entity's complete form
- Gestalt is the German word for shape.
- Brain is holistic, parallel, and analog with self-organizing tendencies
- Appeared in the 1920s

- http://sixrevisions.com/web_design/gestalt-principles-applied-in-design/

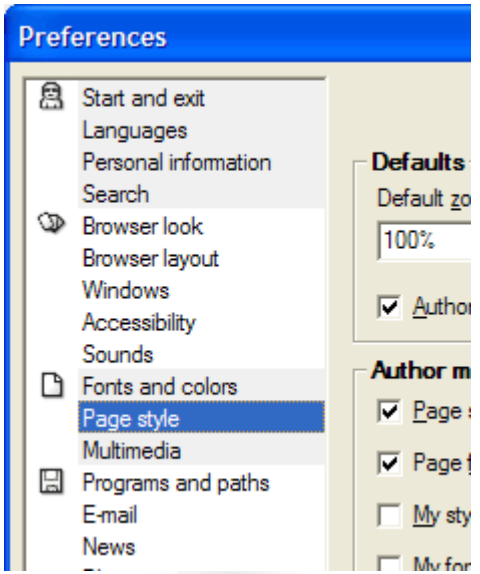
Gestalt theory

“The fundamental formula of Gestalt theory might be expressed in this way. There are wholes, the behavior of which is not determined by that of their individual elements, but where the part-processes are themselves determined by the intrinsic nature of the whole. It is the hope of Gestalt theory to determine the nature of such wholes. With a formula such as this one might close, for Gestalt theory is neither more nor less than this.“

Max Wertheimer, 1925: Über Gestalttheorie, Erlangen, 1925

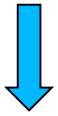
Gestalt theory

- 6 principles related to Gestalt theory:
  - Proximity – the underlying concept is grouping
  - Similarity – we group things perceptually if they appear similar to one another
  - Figure-Ground – stop using busy tiled graphics for our backgrounds – because they took away from the foreground objects
  - Symmetry – the principle of symmetry tells us that when we look at certain objects, we see them as symmetrical shapes that form around their center
  - Common Fate – related items are sharing a “common fate”
  - Closure – we close objects that are themselves not complete

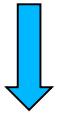


Types of Memory

- Sensory memory (acts as a buffer of perceptions)



- **Short-term memory**
  - temporary, short access time < 0.1 s
  - erased after a few seconds
  - small capacity – 7 chunks, do not overload short-term memory)



- Long-term memory (learning, practicing)
  - Longer access time > 0.1 s, slower erasing, large capacity

UI Design

- Schneiderman's eight Golden Rules of Interface Design [http://www.devirtuoso.com/2009/05/8-golden-rules-of-interface-design]

1. Strive for consistency
2. Enable frequent users to use shortcuts
3. Offer informative feedback
4. Walk user through more complicated tasks
5. Offer simple error handling
6. Permit easy reversal of actions
7. Make the user feel in control
8. Keep it simple

Consistency

- Good user interface design is about getting a user to have a consistent set of expectations, and then meeting those expectations
- Use consistent terminology
- Consistent colors, fonts, icons, etc.

Shortcuts

- This is especially valid for users that use the interface on a regular basis
- Something to consider might be, abbreviations, function keys, hidden commands and automated actions

Feedback

- For every action that the user does, there should be some sort of feedback, either good or bad
- For more frequent and minor actions the response can be minimal

Walk User Through - Navigation

- When you have an action that requires several steps, be sure to separate it into a logical beginning, middle and end
- After each step be sure to give feedback that will clarify that the step was done correctly and they can move on to the next step
- At the end of all the steps be sure to let the user know that they are completed and that they have finished all the requirements

Error Handling

- Try to design the system so the user **cannot make a serious error**
- If an error is made, the system should be able to detect the error and **offer simple, comprehensible solution** for handling the error

Undo

- Give a way for the user to **undo an error**
- This will help keep the user at ease if they know that not everything has to be perfect
- This will **encourage further exploration of your interface**

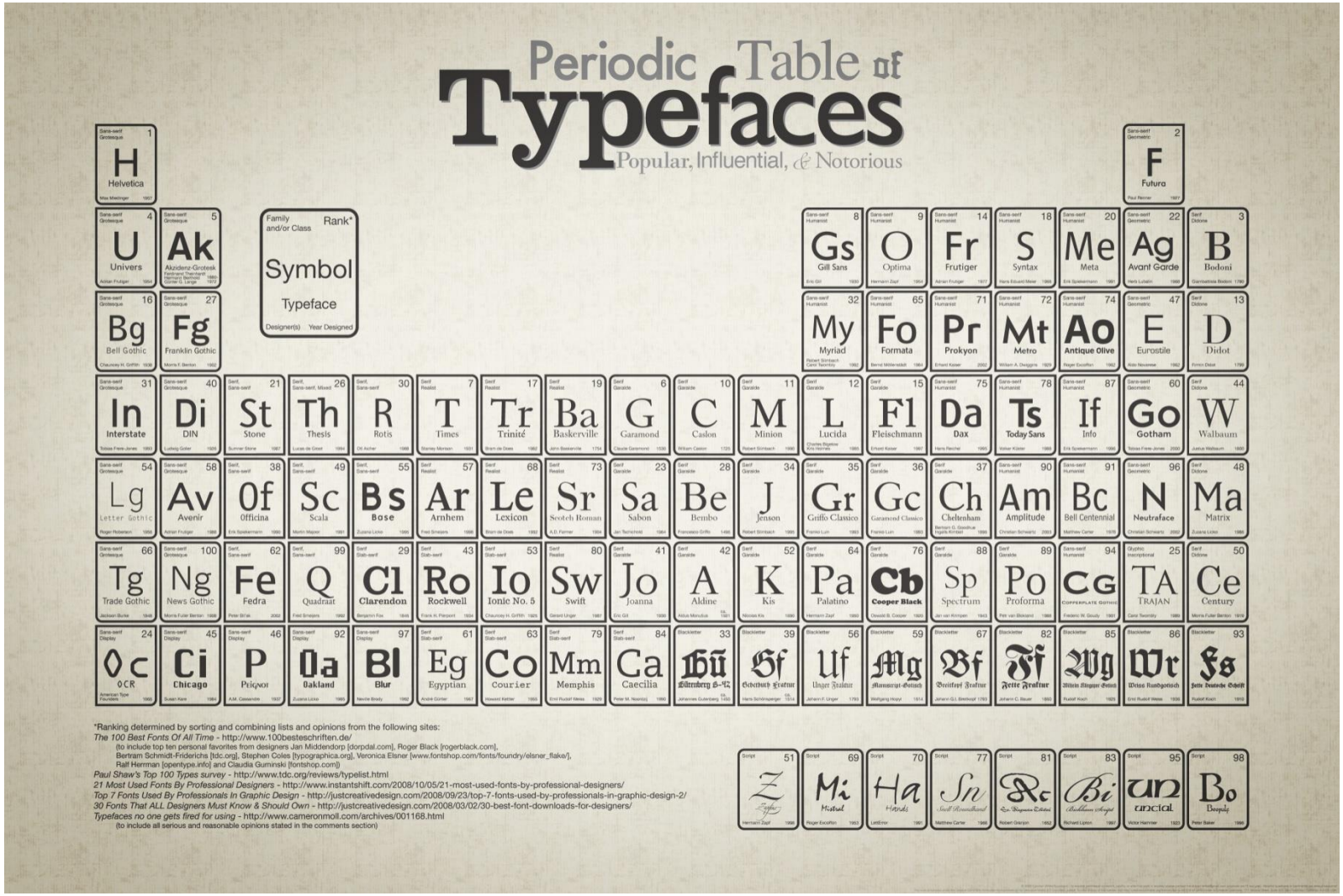
Full Control

- **Experienced users** always want to feel like they are in control of the system
- Make sure the design makes the user feel in control and not just responding to a situation

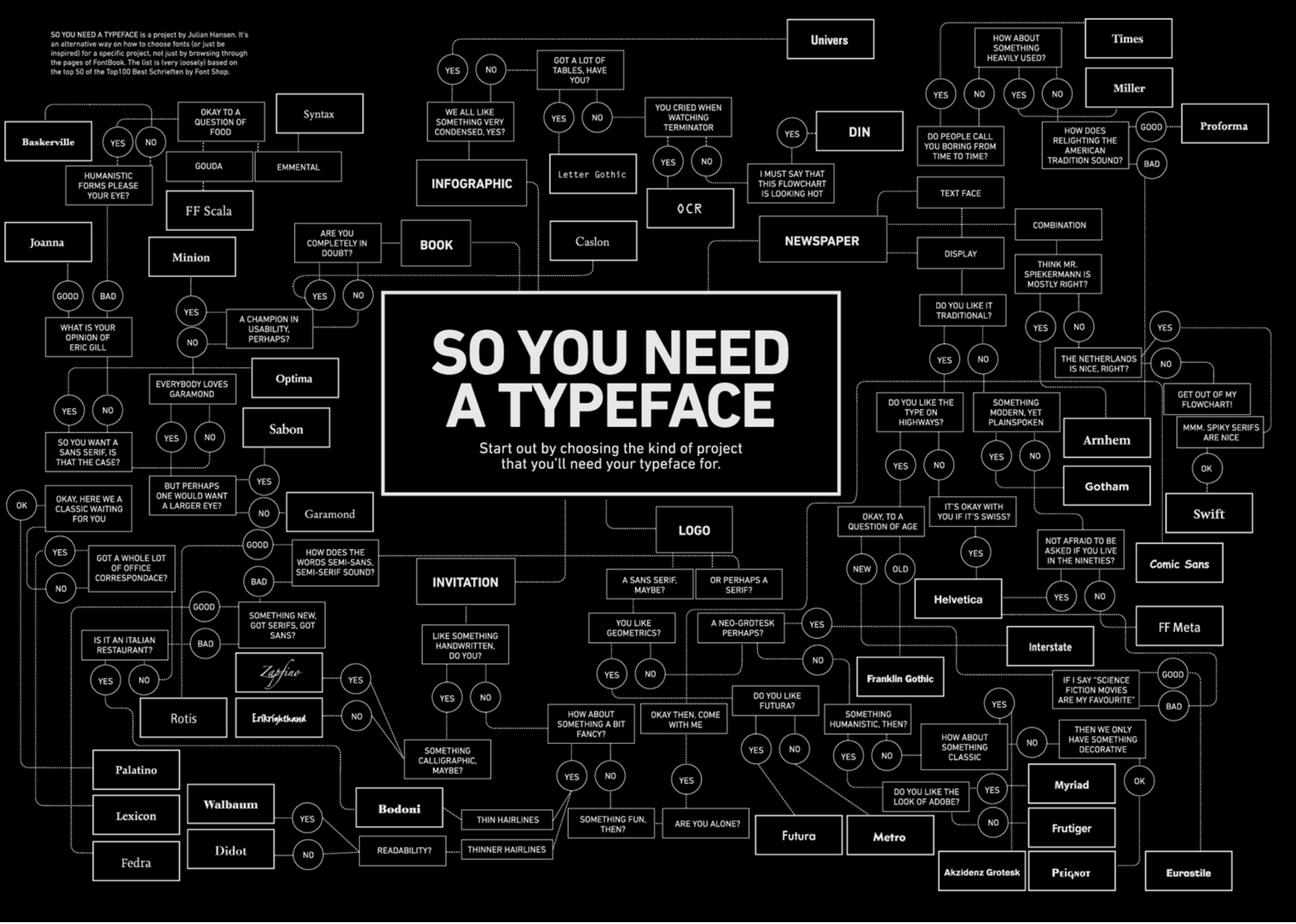
Simple Design

- People have a **limited short-term memory**
- Having to keep track of several things at once can leave a user frustrated or incapable of using your interface
- Try and consolidate multiple pages, reduce unneeded motion, and generally **just keep things simple**

Typefaces



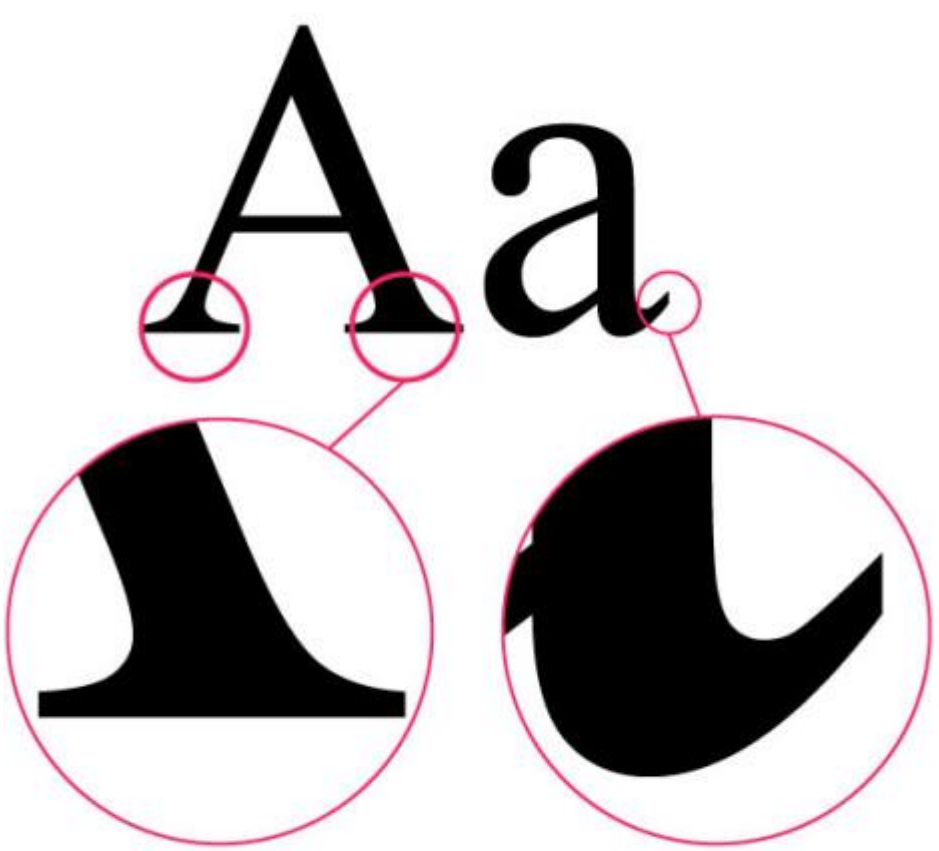
Source: http://squidspot.com/Periodic_Table_of_Typefaces.html


SO YOU NEED A TYPEFACE is a project by Julian Hansen. It's an alternative way on how to choose fonts (or just be inspired) for a specific project, not just by browsing through the pages of FontBook. The list is (very loosely) based on the top 50 of the Top100 Best Schrieften by Font Shop.
SO YOU NEED A TYPEFACE
Start out by choosing the kind of project that you'll need your typeface for.
Univers
Times
Miller
Proforma
DIN
Syntax
Baskerville
Joanna
Minion
Optima
Sabon
Garamond
Palatino
Lexicon
Fedra
Walbaum
Didot
Bodoni
Rotis
Zapfino
Erikrightshand
Futura
Metro
Helvetica
Arnhem
Gotham
Swift
Comic Sans
Interstate
FF Meta
Franklin Gothic
Myriad
Frutiger
Akzidenz Grotesk
Eurostile
Caslon
Letter Gothic
OCR
FF Scala
GOUDA
EMMENTAL
BOOK
INFOGRAPHIC
NEWSPAPER
LOGO
INVITATION
TEXT FACE
COMBINATION
DISPLAY
Source: http://www.julianhansen.com/#/zimmer/

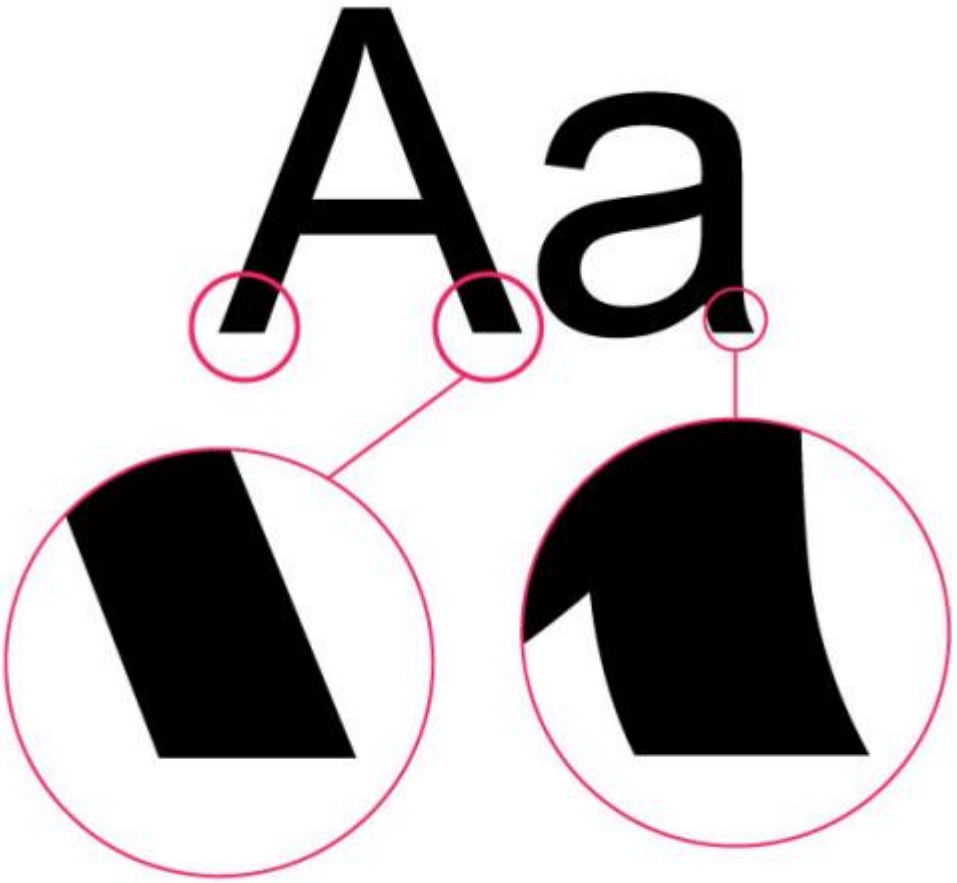
Typefaces vs. Fonts

- **Typography** – the art and technique that consists of arranging type (form) with the purpose of writing
  - The purpose is to make sure the text is easy to read
- **Typeface** – set of typographical symbols and characters (letters, numbers, and other chars)
  - Verdana
- **Type family** – group of typefaces with related design
  - serif, sans-serif, script, display, and so on
- **Font** – defined as a complete character set within typeface of a particular weight, width, and style
  - Verdana 12pt italic

Serif vs. Sans-Serif



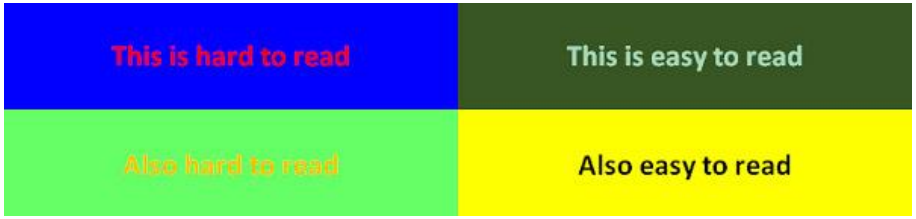
SERIF



SANS SERIF

Basic Principles of Typography

1. Don‘t use too many typefaces (type families)
2. Contrast is good, but the wrong colors can be painful



3. Limited use of display typefaces
   - Complex display typefaces look interesting, but not designed to be used for bodies of text
4. Scannable text is a must
   - Reader should be able to easily scan the text for focus points that peak his interest
5. Don‘t distort typefaces

Basic Principles of Typography

5. Don‘t distort typefaces (cont.)

- Each typeface contains styles and weights that are already properly expanded and condensed
- Do not use the bold and italic buttons in character palettes of the software as they are called “false bold/italic”

Typefaces

Serif	Sans-serif
ABCabc	ABCabc
(Georgia)	(Verdana)

- Use **serif for printed work** because serif fonts are usually **easier to read** than sans-serif fonts
- The convention is to use a serif font for the body of the text. A **sans-serif** font is often used **for headings and captions**

Typefaces

Serif	Sans-serif
ABCabc	ABCabc
(Georgia)	(Verdana)

- Use **serif for printed work** because serif fonts are usually **easier to read** than sans-serif fonts
- The convention is to use a serif font for the body of the text. A **sans-serif** font is often used **for headings and captions**

Typefaces

Serif	Sans-serif
ABCabc	ABCabc
(Georgia)	(Verdana)

- **Lower resolution** can make very small serif characters harder to read
- **Use sans serif** for online work and presentations

Typefaces

Display

(Vineta BT)

Script

(Segoe Script)

- **Display** typeface is unsuitable for body copy and are best reserved for headlines or other short copy that needs attention drawn to it
- **Scripts** are based upon handwriten characters and symbols

Typefaces

Dingbat

♥∂∑ℜ∅

(Symbol)

Dingbat (ornament) is a special typeface used for scientific and mathematical formulas or graphic icons

Typefaces

- Proportional – the space a character takes up is dependent on the natural width of that character
- Monospaced – each character takes up the same amount of space

Source: http://www.noupe.com/essentials/icons-fonts/a-crash-course-in-typography-the-basics-of-type.html

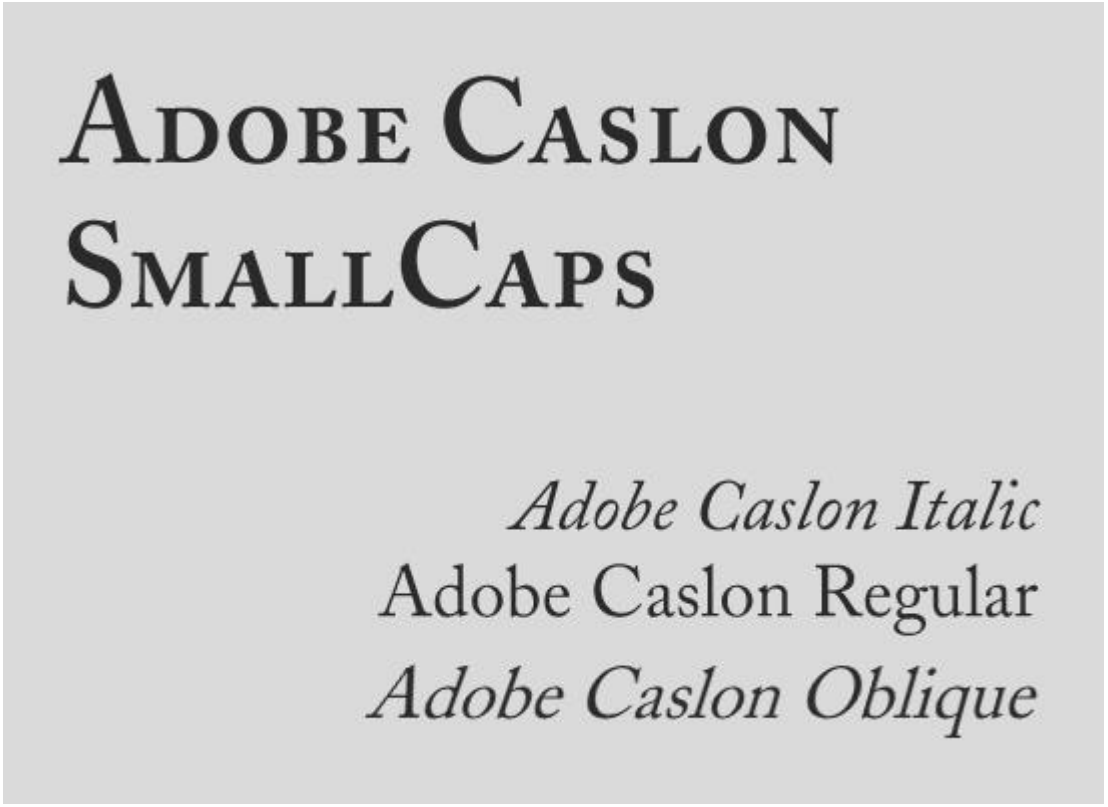
Typefaces

- Weight – refers to the thickness of the strokes that make up the characters

Source: http://www.noupe.com/essentials/icons-fonts/a-crash-course-in-typography-the-basics-of-type.html

Typefaces

- Style – regular, italic, oblique, and small caps



Source: http://www.noupe.com/essentials/icons-fonts/a-crash-course-in-typography-the-basics-of-type.html

Mood of Typefaces

Source: http://www.noupe.com/essentials/icons-fonts/a-crash-course-in-typography-the-basics-of-type.html

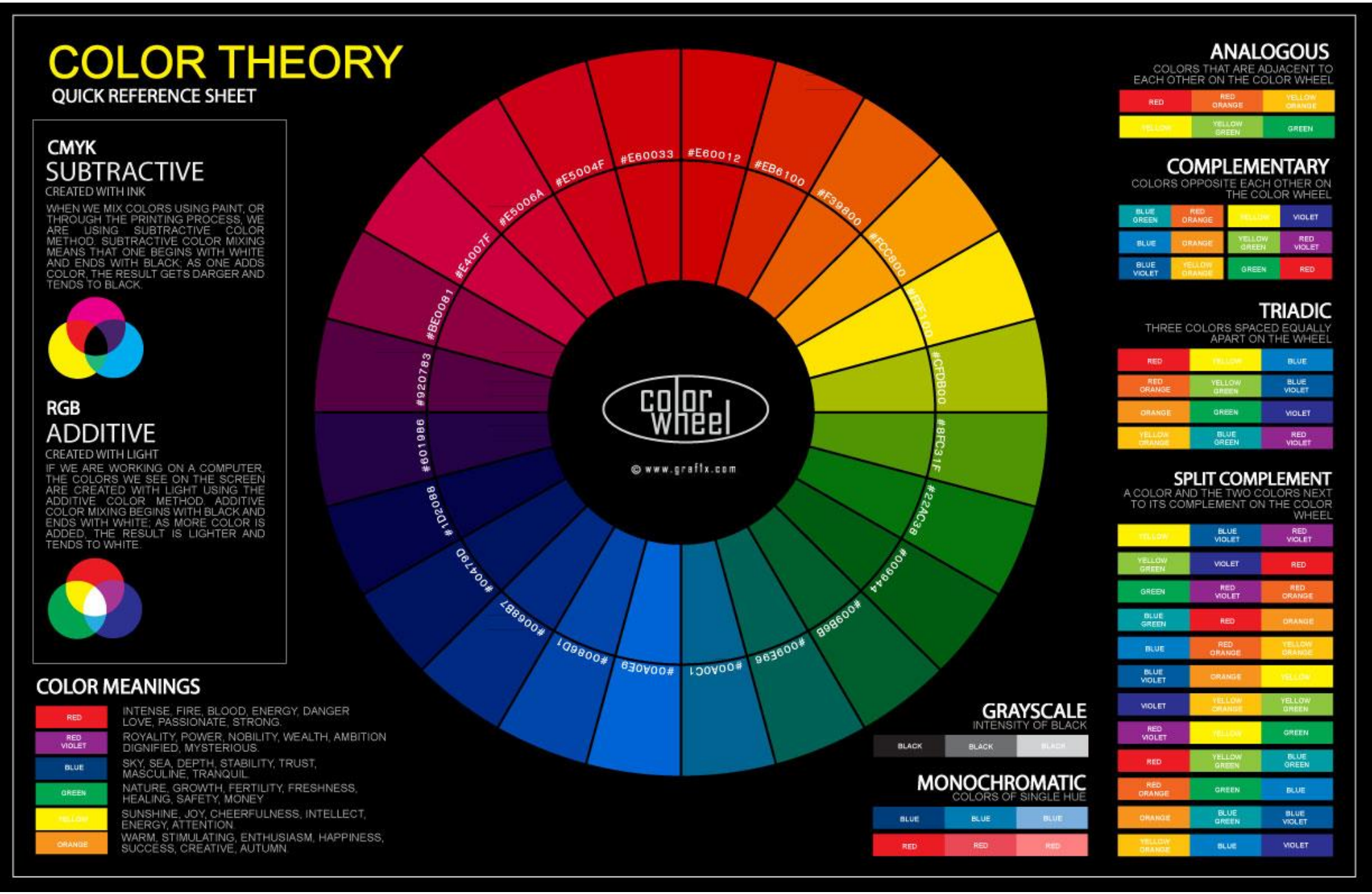
Which Font?

- Times New Roman and Arial are read the fastest

Font Size	Prefered Typeface
10	Verdana
12	Arial
14	Comic Sans

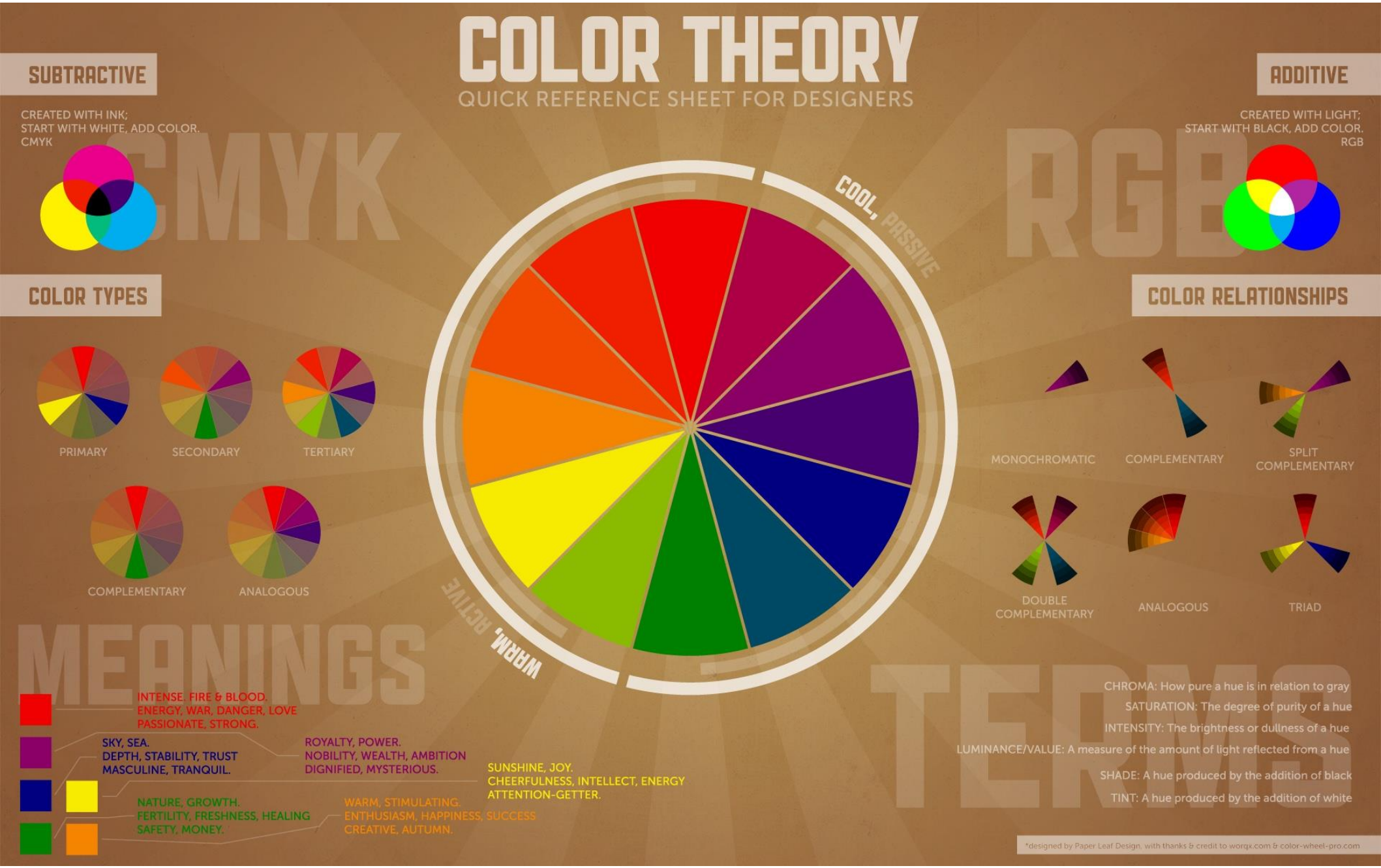
Font Size	Most Legible Typeface
10	Tahoma
12	Courier
14	Arial

Device	Prefered Type Family
Display	Sans Serif
Paper	Serif



Color Wheel

Source: http://graf1x.com/color-wheel-history-and-explanation/



Color Wheel Source: http://paper-leaf.com/blog/2010/01/color-theory-quick-reference-poster/

Colour Harmonies

- **Complementary** – opposite colors on the color wheel; high contrast creates a vibrant look especially at full saturation
- **Split-complementary** – base color + two colors adjacent to its complement

Complementary

Split-Complementary

Colour Harmonies

- **Analogous** – colors that are next to each other; pleasing, harmonious, create serene and comfortable design
- **Triadic** – colors evenly spaced around the color wheel; quite vibrant even if unsaturated

Analogous

Triadic

Colors Recap.

- **Color** - spectral power distribution of wavelengths of light waves reflected from objects
- **Color wheel** - color spectrum bent into a circle
- **Primary colors** - the most basic colors on the color wheel, red, yellow and blue. These colors cannot be made by mixing
- **Secondary colors** - colors that are made by mixing two primary colors together. Orange, green and violet (purple)
- **Tertiary colors** - colors that are made by mixing a primary color with a secondary color

Primary Colors

Secondary Colors

Tertiary Colors

Colors Recap.

- **Hue** - the name of the color
- **Intensity** - the brightness or dullness of a color
- **Color value** - the darkness or lightness of a color (e.g. pink is a tint of red)
- **Tints** - are created by adding white to a color
- **Shades** - are created by adding black to a color
- **Optical color** - color that people actually perceive - also called local color
- **Arbitrary color** - colors chosen by the artist to express feelings or mood

SDI and MDI

- SDI – Single Document Interface
  - Preferred way to go for normal applications
- MDI – Multiple Document Interface
  - Now out of favor
  - Consumes less system resources
  - Useful for professional applications

SDI

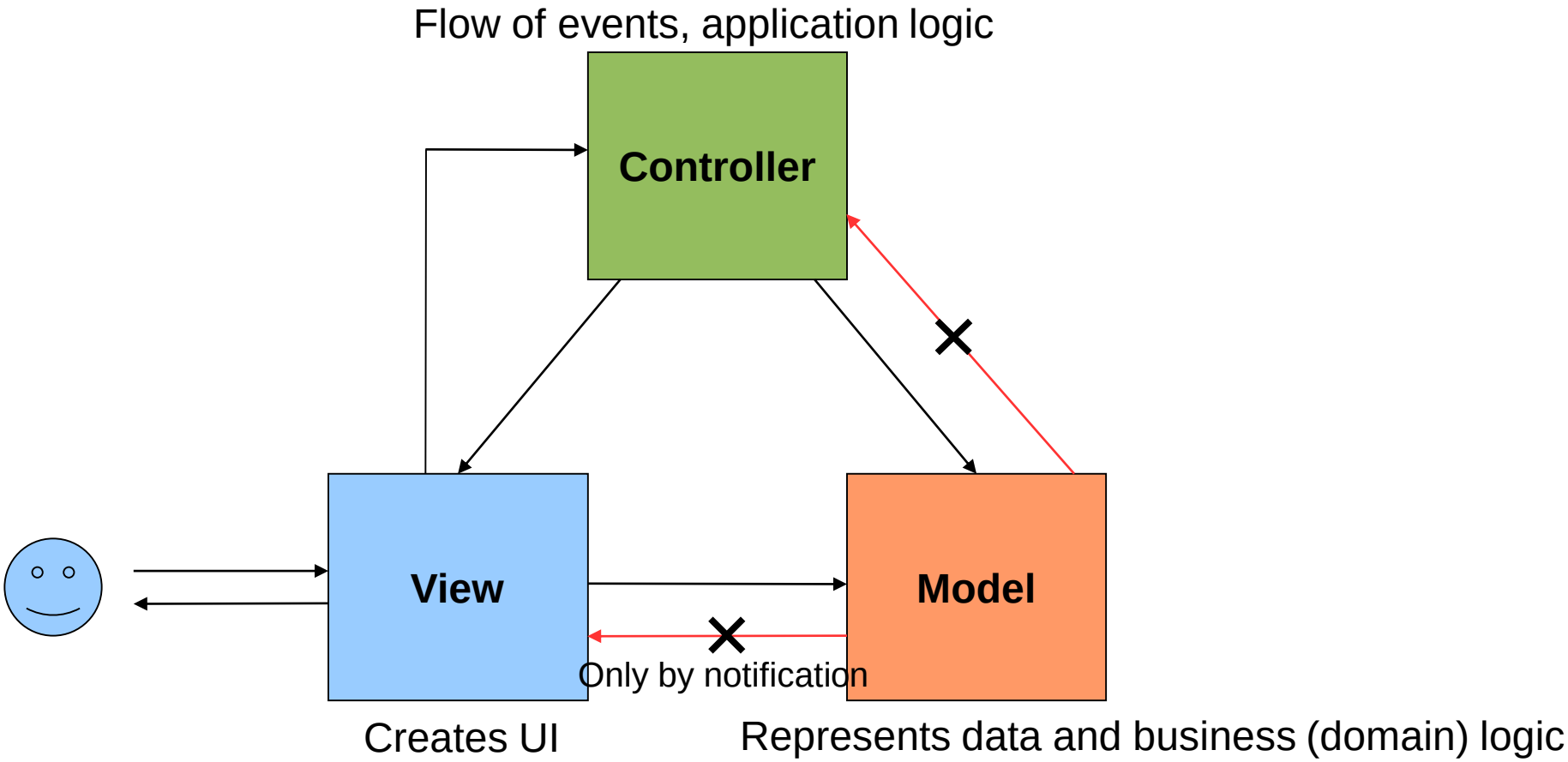
- Each instance has its own set of menus and toolbars. To switch between the documents you use the task bar to switch execution.
- Pros: data centered, less confusing

MDI

- A parent window with children (unlocked and floating)
- Many SDI apps can also act as a MDI app (Excel, Word, Power Point, etc.)
- Child windows share the menus of the parent window
- MDI can present multiple views of the same object or allows comparing two (or more) different object

Model-View-Controller Architecture

- MVC is the domain model of relationships from real word (e.g. reflect business rules)



Java and Swing GUI Toolkit

- AWT Abstract Window Toolkit (import java.awt.*)
  - It's there from the beginning of Java language, aimed at creating complex UI. Intensive use of design patterns (based on Model-View-Controler).
- Swing (**import javax.swing.***)
  - It's the extension of the AWT, contains many brand new components, standard dialog windows, Look&Feel, also based on MVC.
  - Both appearance and behaviour of widgets are implemented in pure Java.

Java and Swing GUI Toolkit

- Layout management – 8 basic types of layout
- Swing is contained in JFC (Java Foundation Classes)
  - Support for data transfers (Cut/Copy/Paste, Drag&Drop)
  - Contains Undo/Redo framework
  - Internationalization, Accessibility (e.g. visually impaired)
- Multiplatform

Java and Swing GUI Toolkit

- Java SE Development Kit (JDK)
  - http://www.oracle.com/technetwork/java/javase/downloads/index.html
- Java SE Technical Documentation
  - http://docs.oracle.com/javase/7/docs/
  - API
    - http://docs.oracle.com/javase/7/docs/api/
  - **Swing**
    - **http://docs.oracle.com/javase/tutorial/uiswing/index.html**
- Editors
  - PSPad, Notepad, NetBeas, Eclipse, …

Java GUI Basics

- Every visible component (widget) must be a descendant of java.awt.Component

  java.lang.Object

  extended by **java.awt.Component**

  extended by java.awt.Container

  extended by javax.swing.JComponent

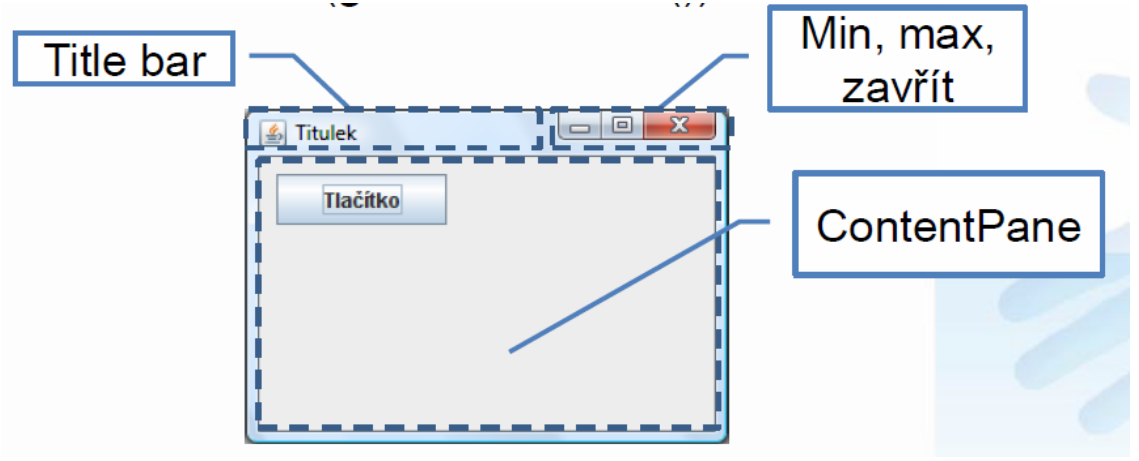
  extended by javax.swing.AbstractButton

  extended by **javax.swing.JButton**

- Swing follows the MVC model almost everywhere (e.g. even button has its own model java.swing.ButtonModel)

Java GUI Basics

- JFrame (remaining two main components are JDialog and JApplet)
  - Main window of every application
  - Communicate with OS
  - Container for other components (JButton, JLabel, JPanel) is available through getContentPane() method)



Java GUI Basics

- Containers:
  - JPanel, JTabbedPane, JSplitPane, JScrollPane, …
  - Facilitate the placement of other components
- Atomic components:
  - JLabel, JButton, JComboBox, JTextFiled, JTable,...
  - Enable interaction with the user

How to create single button

- 1. Instantiate

```
JButton btn = new JButton(“my button“);
```

- 2. Configure

```
btn.setPrefferedSize(new Dimension(100, 20));
btn.setText(“MY BUTTON”);
```

- 3. If we deal with container, we can add some descendants here.
- 4. Otherwise we can add our component in the container.

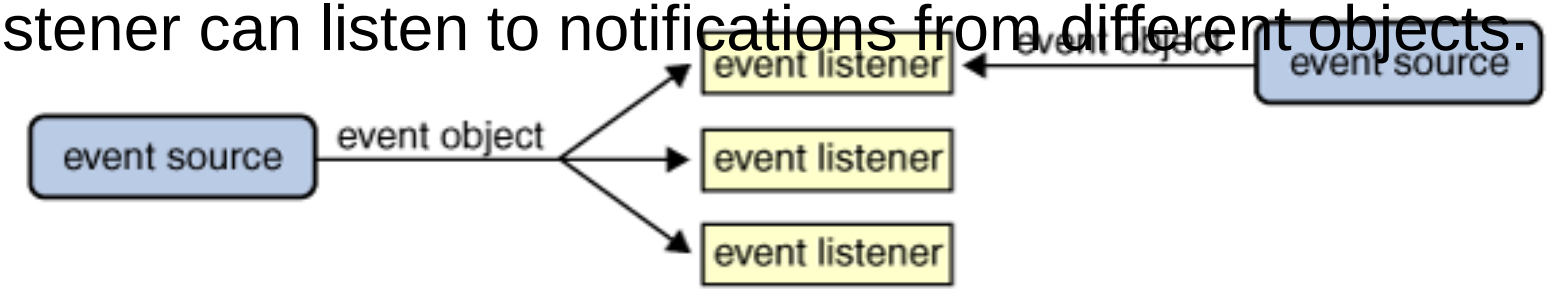
```
panel.add(btn);
```

- 5. Listeners registration

```
btn.addXXXListener(listener);
```

Event Listener

- Event source
  - An object generating events
  - Maintain the list of registered listeners:
    `add<Something>Listener()`
    `remove<Something>Listener()`
- Listener
  - Multiple listeners can register to be notified of events of a particular type from a particular source. Also, the same listener can listen to notifications from different objects.



Bootstrap

- Bootstrap is a CSS framework for development of Web application and Web pages
- Standardized way for consistent typography, form layouts, and common widget appearance
- Support for responsive design across a wide range of web browsers and devices (from handhelds with small screens to large desktop displays)

Bootstrap

- Important links
- Main page with installation instructions

  https://getbootstrap.com
- Extensive documentation and examples

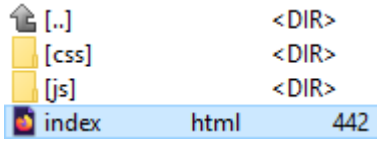
  https://getbootstrap.com/docs/5.3/getting-started/introduction

  https://getbootstrap.com/docs/5.3/examples/

Bootstrap Minimum HTML Page

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1, shrink-to-fit=no">
    <link rel="stylesheet" href="css/bootstrap.min.css">
    <title>Hello, world!</title>
  </head>
  <body>
    <div class="container">
      <h1>Hello, world!</h1>
    </div>
  </body>
</html>
```

We assume the following organization of html page file and the Bootstrap library



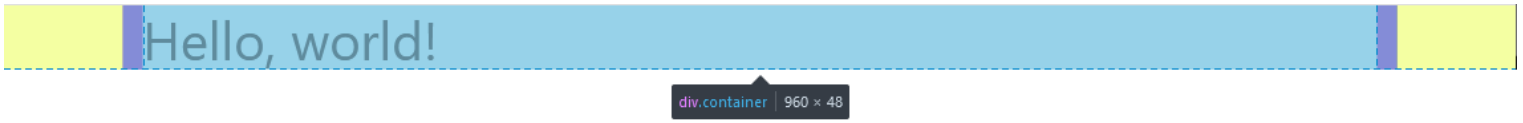
https://github.com/twbs/bootstrap/releases/download/v5.3.3/bootstrap-5.3.3-dist.zip

Bootstrap Containers

- Containers are the most basic layout element in Bootstrap and are required when using default grid system
- Containers are used to contain, pad, and (sometimes) center the content within them.
- Containers can be nested (but not necessary most of time)

```
<div class="container">
  <h1>Hello, world!</h1>
</div>
```

Default container class is a responsive, fixed-width container, meaning its max-width changes at each breakpoint (see the next slide)



Bootstrap Containers

- Bootstrap comes with three different containers
- `container` sets a max width at each responsive breakpoint
- `container-fluid` spanning the entire width of the viewport
- `container-{breakpoint}` 100% width until the specified breakpoint

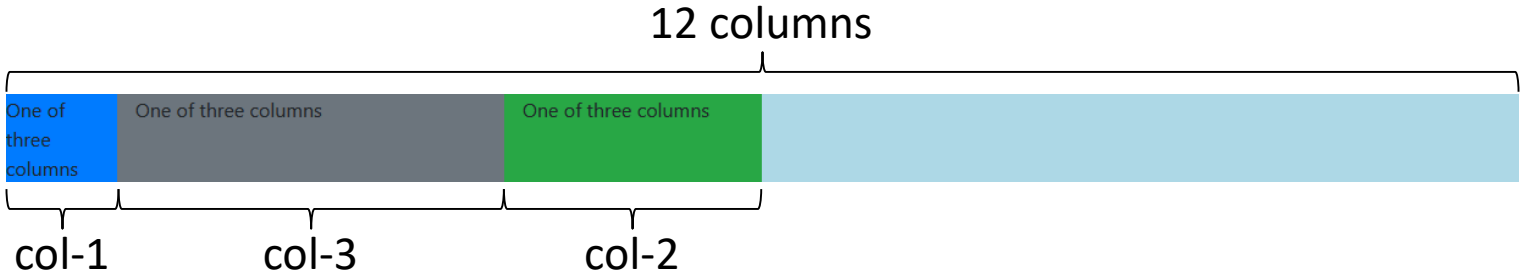
	Extra small <576px	**Small** ≥576px	**Medium** ≥768px	**Large** ≥992px	**Extra large** ≥1200px
`.container`	100%	540px	720px	960px	1140px
`.container-sm`	100%	540px	720px	960px	1140px
`.container-md`	100%	100%	720px	960px	1140px
`.container-lg`	100%	100%	100%	960px	1140px
`.container-xl`	100%	100%	100%	100%	1140px
`.container-fluid`	100%	100%	100%	100%	100%

Bootstrap Grids

- Bootstrap’s grid system uses a series of containers, rows, and columns to layout and align content
- Rows are wrappers for columns
- Each column has horizontal padding (called a gutter) for controlling the space between them. This padding is then counteracted on the rows with negative margins. This way, all the content in your columns is visually aligned down the left side
- In a grid layout, content must be placed within columns and only columns may be immediate children of rows
- Grid columns without a specified width will automatically layout as equal width columns (e.g. four instances of col-sm will each automatically be 25% wide from the small breakpoint and up)

Bootstrap Grids

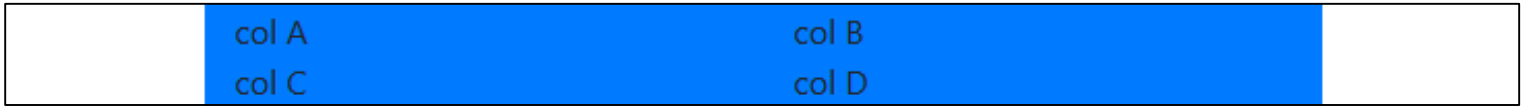
- Maximum number of columns in a single row is 12
- Column classes indicate the number of columns you'd like to use out of the possible 12 per row. If you want three equal-width columns across, you can use col-4 (i.e. number 4 represents the columnspan parameter)




```
<div class="container-fluid" style="background-color: lightblue;">
  <div class="row">
    <div class="col-1 bg-primary">One of three columns</div>
    <div class="col-3 bg-secondary">One of three columns</div>
    <div class="col-2 bg-success">One of three columns</div>
  </div>
</div>
```

Bootstrap Grids

- Create equal-width columns that span multiple lines by inserting a w-100 where you want the columns to break to a new line




```
<div class="container bg-primary">
  <div class="row">
    <div class="col">col A</div>
    <div class="col">col B</div>
    <div class="w-100"></div> <!-- break line -->
    <div class="col">col C</div>
    <div class="col">col D</div>
  </div>
</div>
```

Bootstrap Grids

- Use justify-content-{breakpoint}-{start, center, end, around, between} to align columns horizontally


```
<div class="container bg-secondary">
  <div class="row justify-content-lg-start ">
    <div class="col-lg-2 bg-primary">col A</div>
    <div class="col-md-auto bg-warning">col B</div>
    <div class="col-lg-1 bg-primary">col C</div>
  </div>
</div>
```


```
<div class="container bg-secondary">
  <div class="row justify-content-lg-center">
  …
  </div>
</div>
```

Bootstrap Grids

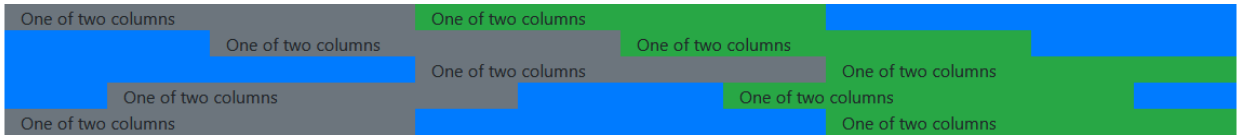
- The gutters between columns in our predefined grid classes can be removed with no-gutters (g-0)


```
<div class="container bg-secondary">
  <div class="row justify-content-lg-start g-0">
    <div class="col-lg-2 bg-primary">col A</div>
    <div class="col-md-auto bg-warning">col B</div>
    <div class="col-lg-1 bg-primary">col C</div>
  </div>
</div>
```

- This removes the negative margins from row and the horizontal padding from all immediate children columns

Bootstrap Grids

- Horizontal alignment




```
<div class="container bg-primary">
  <div class="row justify-content-start">
    <div class="col-4 bg-secondary">One of two columns</div>
    <div class="col-4 bg-success">One of two columns</div>
  </div>
  <div class="row justify-content-center">
    <div class="col-4 bg-secondary">One of two columns</div>
    <div class="col-4 bg-success">One of two columns</div>
  </div>
  <div class="row justify-content-end">
    <div class="col-4 bg-secondary">One of two columns</div>
    <div class="col-4 bg-success">One of two columns</div>
  </div>
  <div class="row justify-content-around">
    <div class="col-4 bg-secondary">One of two columns</div>
    <div class="col-4 bg-success">One of two columns</div>
  </div>
  <div class="row justify-content-between">
    <div class="col-4 bg-secondary">One of two columns</div>
    <div class="col-4 bg-success">One of two columns</div>
  </div>
</div>
```

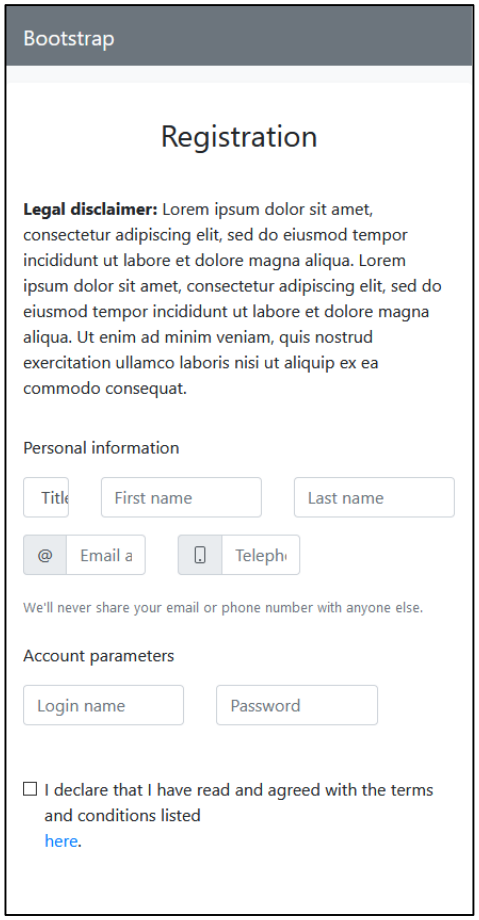
Bootstrap Grids

- Horizontal and vertical padding

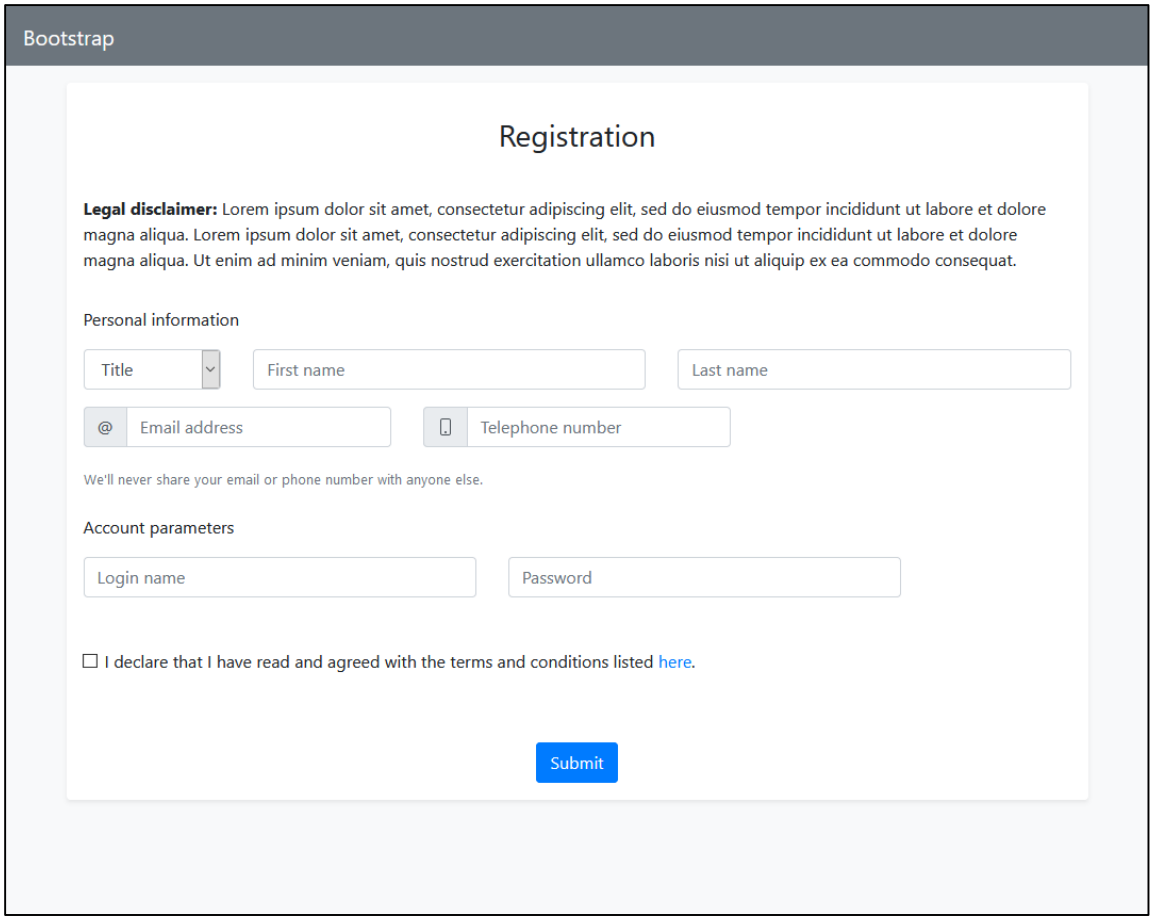

```
<div class="container px-lg-5">
  <div class="row mx-lg-n5">
    <div class="col py-5 px-lg-5 bg-secondary">Custom column padding</div>
    <div class="col py-3 px-lg-5 bg-success">Custom column padding</div>
  </div>
</div>
```

Bootstrap Examples

- Bootstrap is responsive by default



Small display



Large display

Bootstrap Links

- Exhaustive description with examples how the grid system in Bootstrap works may be found here (the most important sections are Layout, Content, Components, and Utilities)

  https://getbootstrap.com/docs/4.4/layout/grid/

- Also see the example below for a better idea of how it all works

  http://mrl.cs.vsb.cz/people/fabian/uro/p3_hints.zip

- Try to experiment with various configurations and explore the consequences

Bootstrap Debug

- You can debug the HTML/CSS code in a web browser by pressing Ctrl+Shift+I (Firefox) or Ctrl+Shift+C (Chrome)