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

Automotive User Interfaces

460-2048

Spring 2025

Last update 7. 4. 2025

Automotive User Interfaces

- Staff List
  - 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

Course web site
mrl.cs.vsb.cz/people/fabian/ura_course.html

Course Prerequisites

- Basics of programming (Python and C++)

- Previous courses (not mandatory)
  - Fundamentals of Programming
  - Fundamentals of Computer Graphics

Main Topics

- Theory and Principles of the design of the interface between man and machine (HMI)
- The types of computer systems inside a car
- Digital cockpit - user interface (HCI), information and control systems
- Text input and output during driving, usage scenarios
- Interface for navigation (3D navigation, visualization of the vehicle's surroundings) and cooperative driving
- Biometric sensors as components of the user interface of the automobile
- 3D graphics and augmented reality (head-up display)
- Infotainment interfaces in the vehicle
- Software tools for the design and implementation of graphical user interfaces for embedded devices
- General user interfaces used outside cars
- Embedded platforms with graphical output (e.g. Nvidia Drive CX/PX, Tegra X1, Jetson TX1) and runtime environments

Automotive User Interfaces

- Grading Policy
  - Complete three assignments from labs

    P1: Design and implementation of a **desktop application** in Python + Tk/Qt [10 p, min 5 p]; application for controlling and visualization of various processes and measurements

    P2: Web UI - Building **responsive web page** with Bootstrap [10 p, min 5 p]; simple responsive web form with various widgets

    P3: Detailed design of a car **dashboard** with following implementation in C++ [25 p, min 15 p]; you can be inspired by an existing look of a modern virtual cockpit

    Projects are quite open ended. There are several correct solutions. Deadlines are given on the course website.
  - **Final exam**
    - Test during the last week [55 p, min 6 p]

Study Materials and References

- Meixner, G., & Müller, C. (2017). *Automotive user interfaces*. Springer: Cham, Switzerland.
- Rümelin, S. (2014). The cockpit for the 21st century (Doctoral dissertation, lmu).
- Wu, H., & Shou, S. (2011). Automotive Cockpit Design 2020.
- https://docs.python.org/3/tutorial
- https://docs.python.org/3/library/tkinter.html
- https://wiki.qt.io/Qt_for_Beginners
- https://doc.qt.io/qt3dstudio/index.html

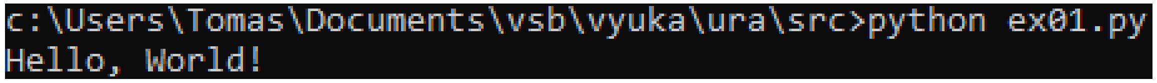
Python Crash Course

- Python is dynamically typed, interpreted, object-oriented, high-level language with automatic memory management which works on many platforms
- Visit www.python.org
  - Downloads Python 3.13.x x64
  - Tutorials
    docs.python.org/3/tutorial/index.html
  - Language reference
    docs.python.org/3/reference/index.html
  - Library reference
    docs.python.org/3/library/index.html

Python Crash Course

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

__author__    = "Tomas Fabian"
__copyright__ = "(c)2025 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

```python
# match-case statement
# introduced in Python 3.10
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")
```

```python
#  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

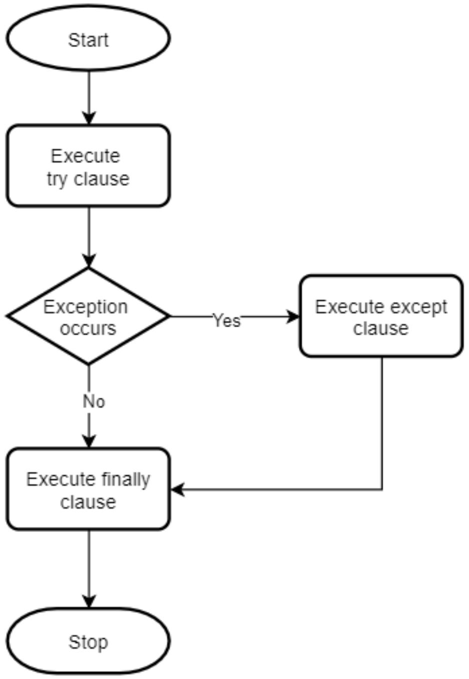
```
# abstract data types in Python
# list
lst = [1, 2, 3, "hi there"]
len(lst)
output: 4
print(type(lst[3]))
output: <class 'str'>
lst[1] = 22 # list is mutable
print(lst)
output: [1, 22, 3, "hi there"]

# 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]

# ternary operator
result = "even" if n % 2 == 0 else "odd"

# 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)

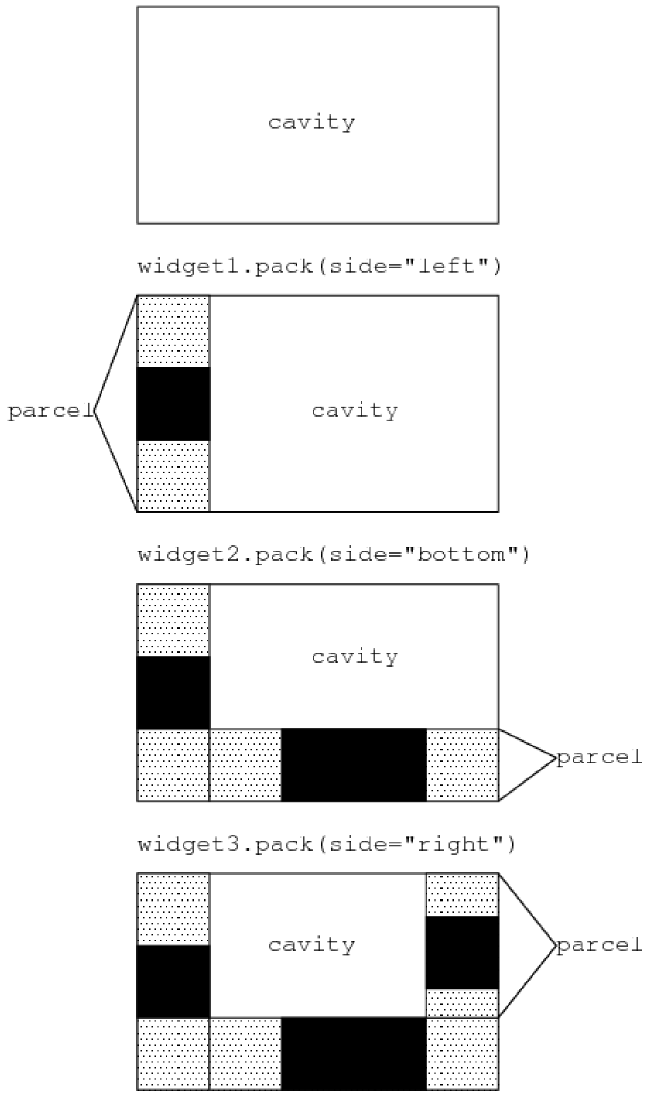
```
•  import tkinter as tk      #imports the entire Tk package

•  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=self.master.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)

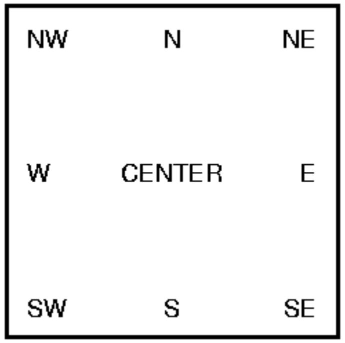
- *Window* – **rectangular area** somewhere on the screen
- *Top-level window* – a window that exists **independently** on the screen
- *Widget* – generic therm for any **building block** that make up a GUI
- e.g. Button, Label, Entry, Frame
- *Frame* – basic widget that can **contain** other widgets.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

Layout management

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


cavity
widget1.pack(side="left")
parcel
cavity
widget2.pack(side="bottom")
cavity
parcel
widget3.pack(side="right")
cavity
parcel

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

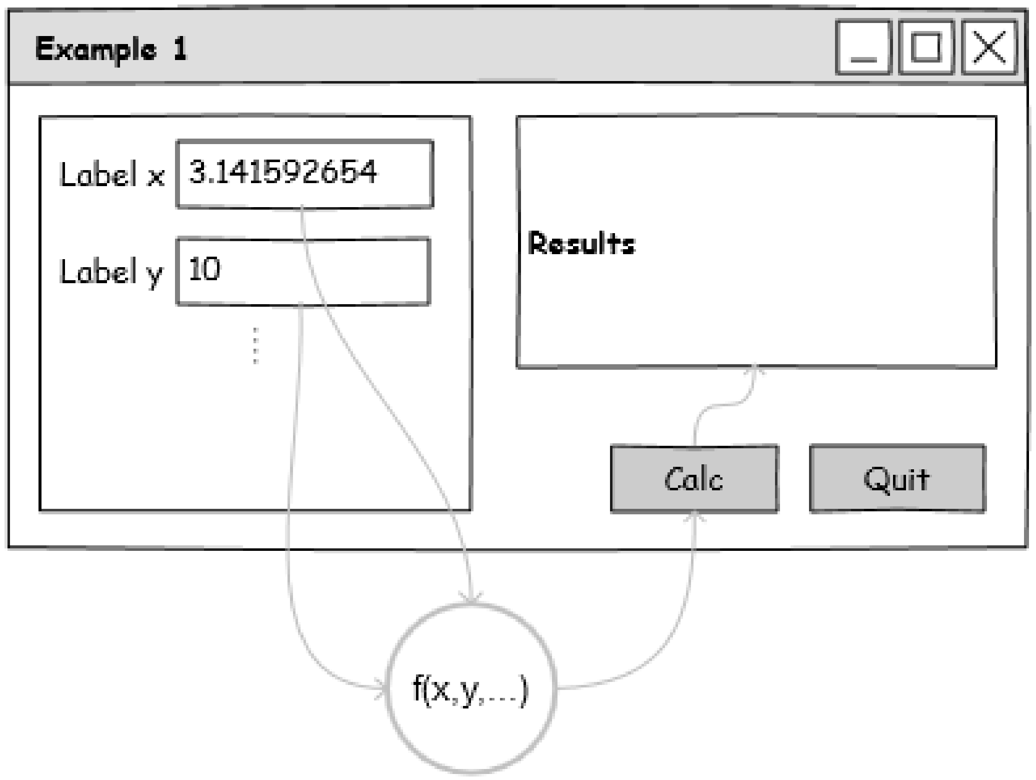
- Widgets are arranged in a window by three different geometry managers
- **PACK** .pack()
  - **fill** – fill any empty space in ‘x ’, ’ y ’, ’ both ’, ’ none ’
  - **ipad**{**x**,**y**} – designating internal padding on each side of the slave widget
  - **side** – ‘left’, ‘right’, ‘top’, ‘buttom’
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

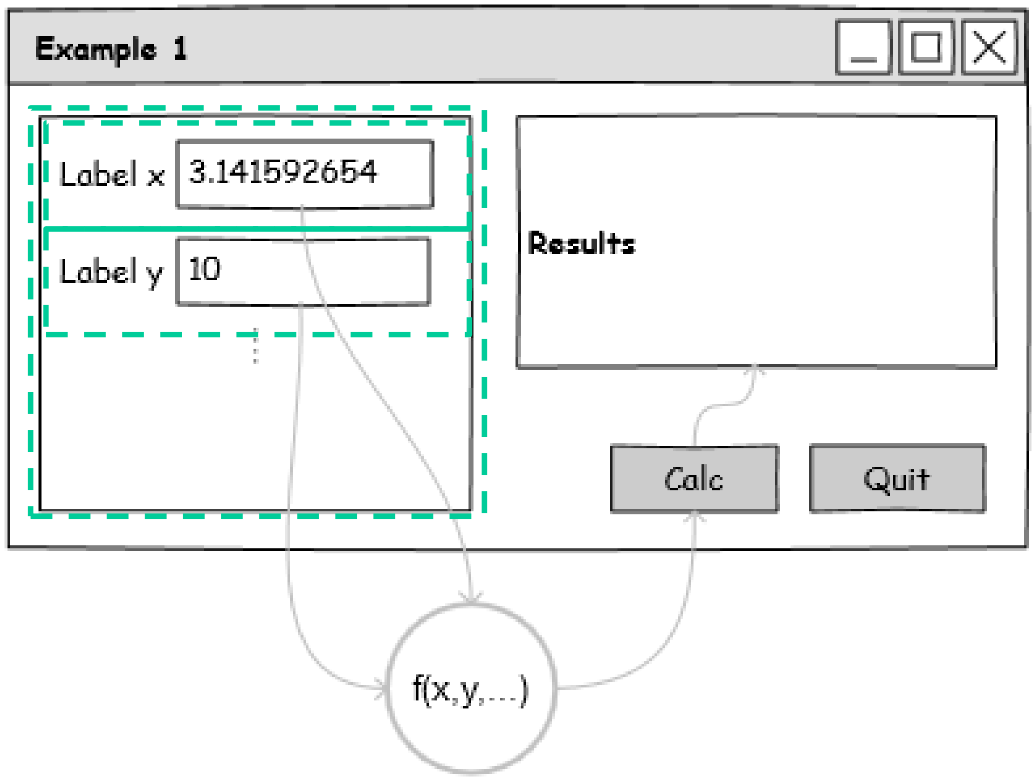
Example 1

- 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

```
• try:
•     x = float(self.x.get())
•     print("y=" + str(x))
• except ValueError:
•     tk.messagebox.showwarning("Value Error", "Real number required.")
•     self.x.set("")
```

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_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

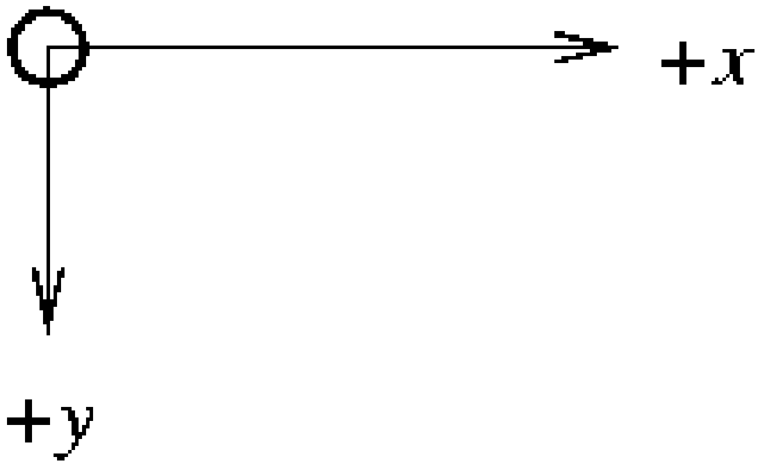
```
self.reds = []

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()))
```

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")`

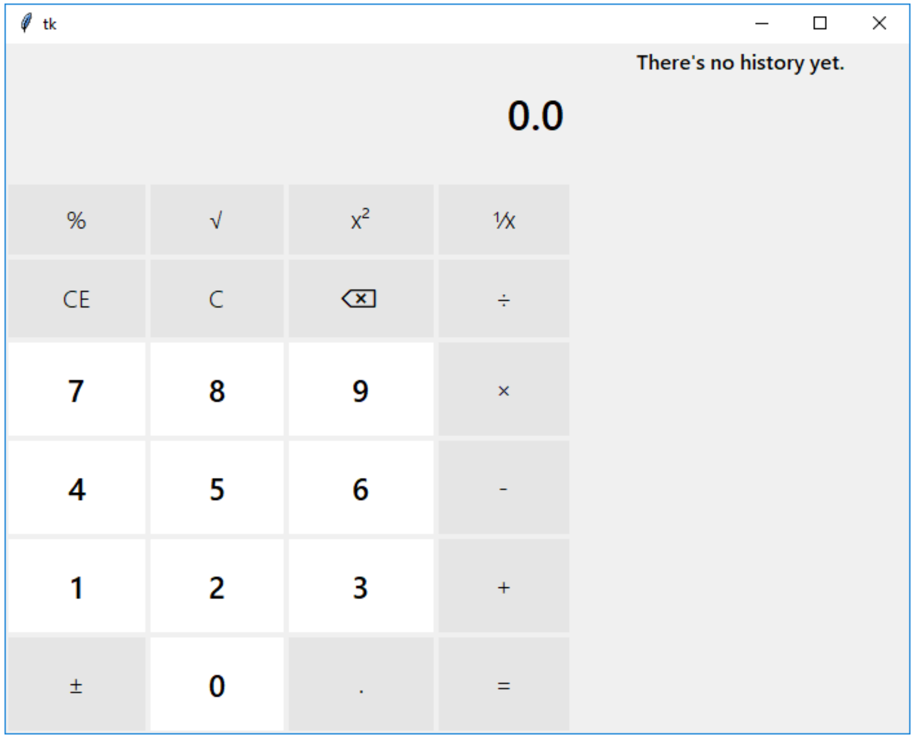
The coordinate system

- 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.



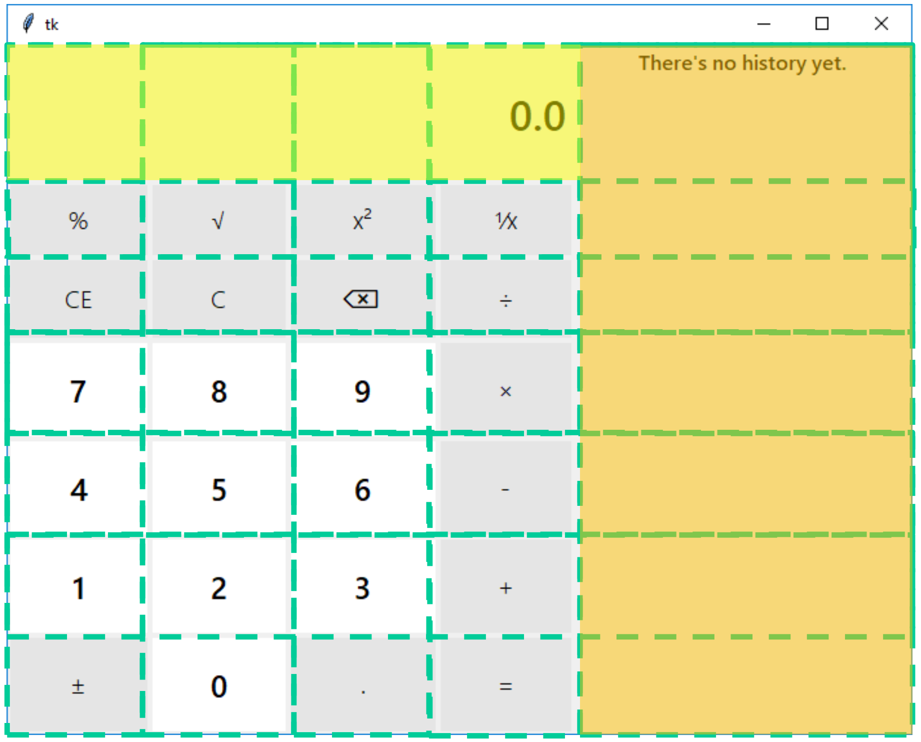
Example 2

- 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 2

columnspan = 4



rowspan = 7

Customized Look and Behavior

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"))

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`

Font Settings

In Windows 11, the Segoe Fluent Icons font is replaced with Segoe MDL2 Assets as the recommended symbol icon font. **Segoe MDL2 Assets** is still available, but we recommend updating to the new **Segoe Fluent Icons**.

# 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

Colors

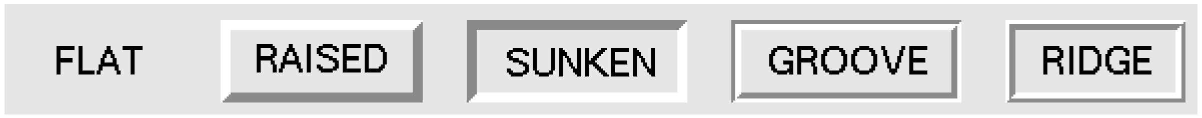
- There are two ways to specify colors:
  - You can use a string specifying the proportion of red, green, and blue in hexadecimal digits:

#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", ...).

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)`

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'))`

Bindings and Events

- Allows you to watch for certain events
- `widget.bind(event, handler)`
- Handler (callback) function trigger when event occurs

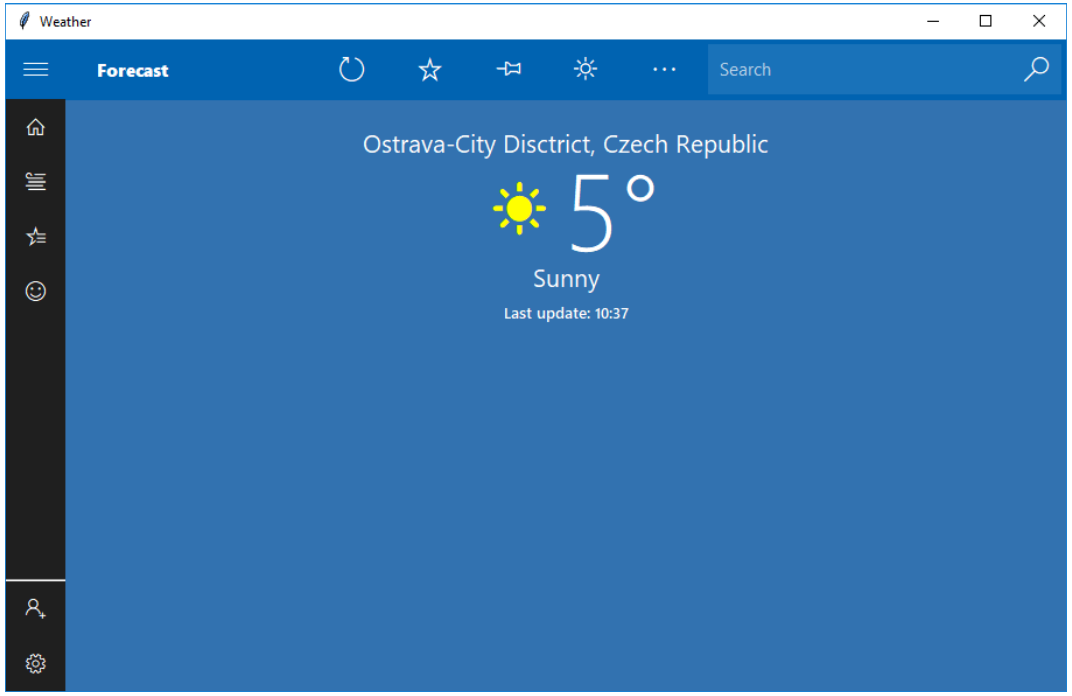
-
  ```
  def click(event):
   print 'You have just clicked on
     {0}'.format(event.widget['text'])
  ```
- `btn = Button(text='Press me')`
- `btn.pack()`
- `btn.bind('<Button-1>', click, [add='' or '+'])`
- Add is optional, either '' or '+'. Passing an empty string denotes that this binding is to 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>`
- 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.

Example 3

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

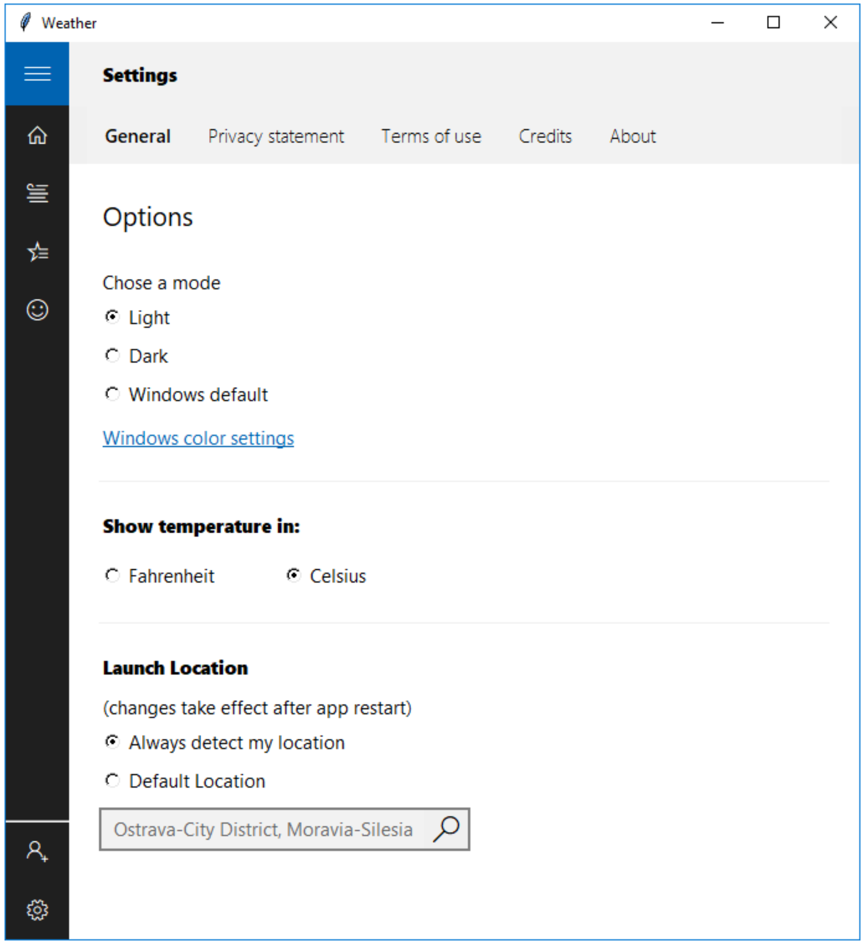


Ostrava-City Disctrict, Czech Republic

5°

Sunny

Last update: 10:37



Options

Chose a mode

[✓ Light]

Dark

Windows default

Windows color settings

Show temperature in:

Fahrenheit [✓ Celsius]

Launch Location

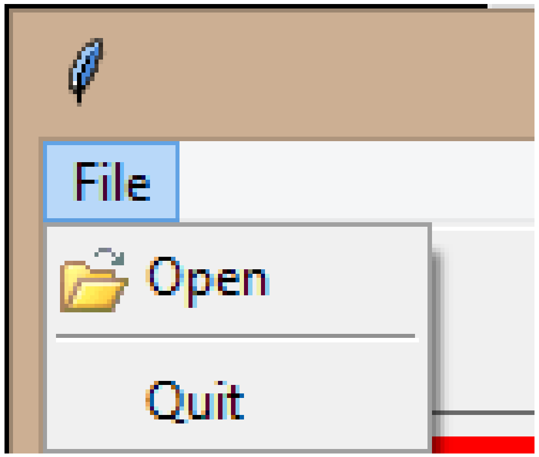
(changes take effect after app restart)

[✓ Always detect my location]

Default Location

Ostrava-City District, Moravia-Silesia

Menu

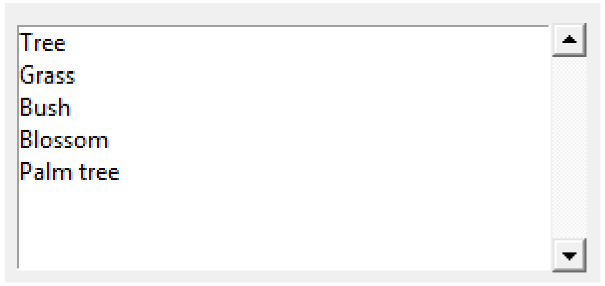


```
self.menubar = Menu(self.master)
self.master.config(menu=self.menubar)

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()
```

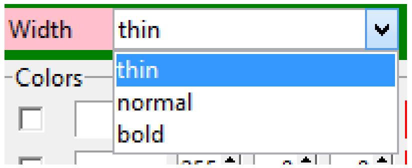
Listbox

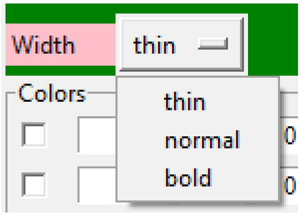


```
self.v = StringVar()
self.v.set("Tree Grass Bush Blossom")

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





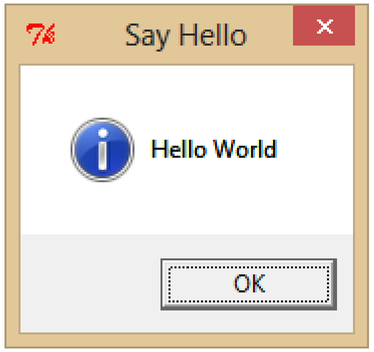
```
from tkinter import ttk
…
frame_width = tk.Frame(frameleft, bg="green")
frame_width.pack(side="top", fill="x")
label_width = tk.Label(frame_width, width=7, anchor="w",
bg="pink", text="Width")
label_width.pack(side="left", pady="5")
brush_width = tk.StringVar(root)
brush_width.set("thin") # default 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)
```

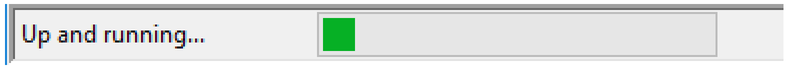
`Or "determinate"`

Determinate:

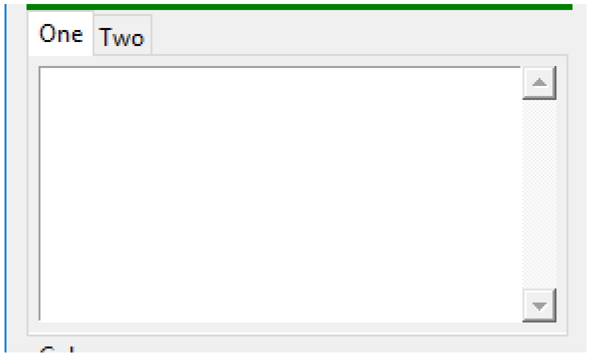
```
self.work = tk.IntVar()
self.work.set(75)
self.progress = ttk.Progressbar(self.frameStatus,
orient=tk.HORIZONTAL, length=200, mode="determinate",
maximum=100, variable=self.work)
```

Indeterminate:

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

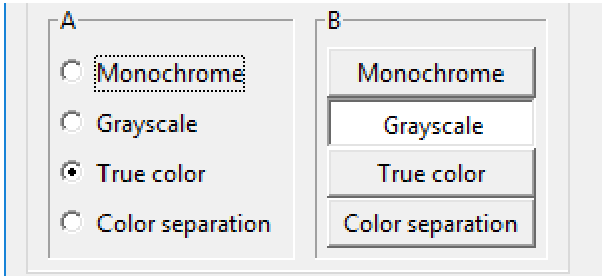


Tabbed Panes



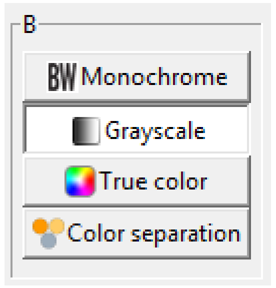
```
import tkinter as tk
from tkinter import ttk
from tkinter.scrolledtext import ScrolledText

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:
  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)
```

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)`

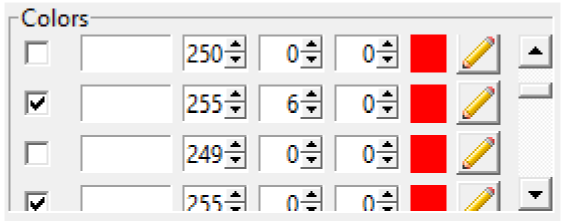
Status bar

- `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)



☐		250	0	0	🟥
✓		255	6	0	🟥
☐		249	0	0	🟥
✓		255	0	0	🟥

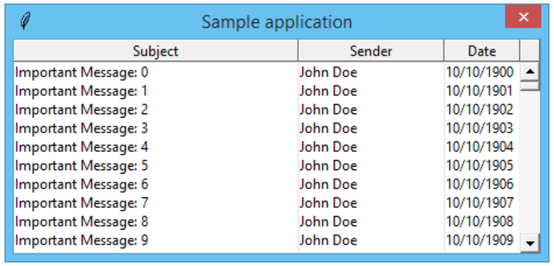
Scrollable Frame 2/2

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

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()))
```

Tables



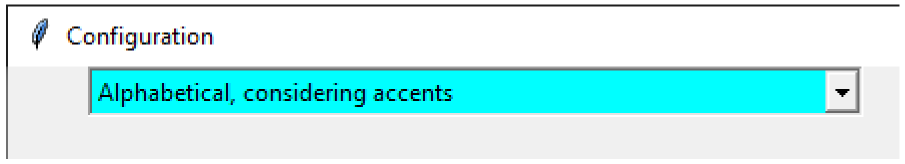
Subject	Sender	Date
Important Message: 0	John Doe	10/10/1900
Important Message: 1	John Doe	10/10/1901
Important Message: 2	John Doe	10/10/1902
Important Message: 3	John Doe	10/10/1903
Important Message: 4	John Doe	10/10/1904
Important Message: 5	John Doe	10/10/1905
Important Message: 6	John Doe	10/10/1906
Important Message: 7	John Doe	10/10/1907
Important Message: 8	John Doe	10/10/1908
Important Message: 9	John Doe	10/10/1909

```
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)
```

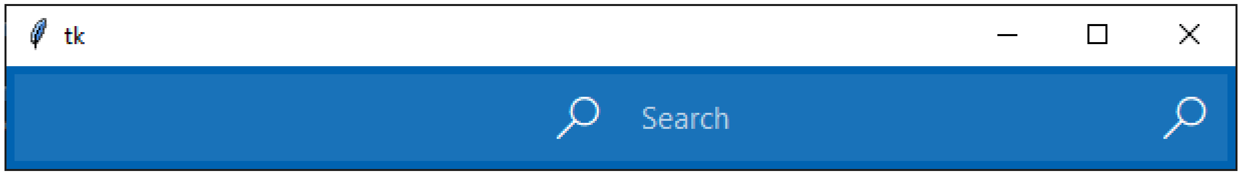
Download the multilistbox.py file from my web site and copy it into your project's folder.

Hints for Python/Tk Assignments

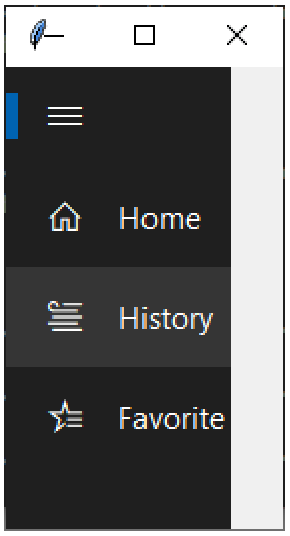
- Source codes with implementations of selected GUI elements in Python/Tk are on the website of this course in the tk_hints.zip file



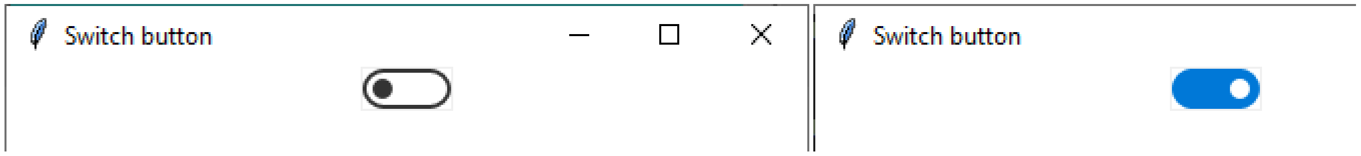
combobox.py



search.py



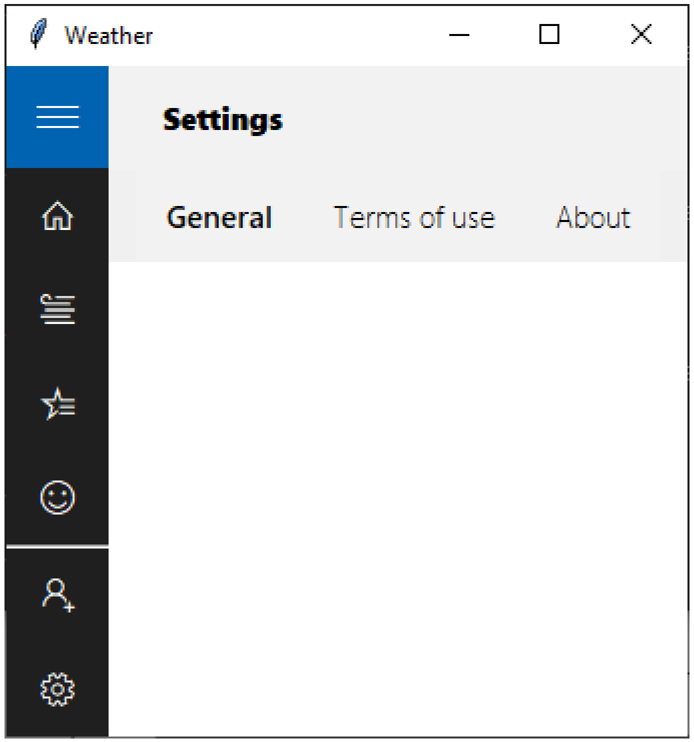
storelikemenu.py



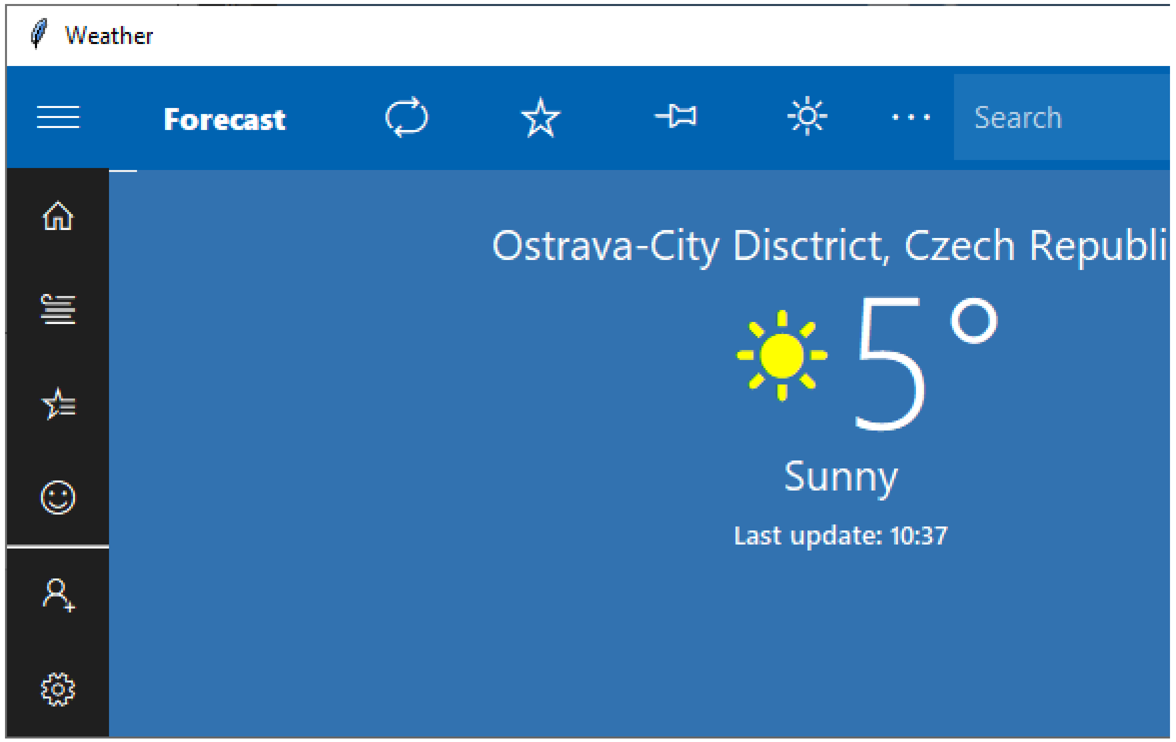
switchbutton.py

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 tk_hints.zip file



storelike.py



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")`

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

UI Mockups Tools

- Pencil – a free GUI prototyping tool for various desktop platforms
  https://pencil.evolus.vn/
- MS Visio – medium-fidelity mockups of software applications (wireframe template)
- Mockplus Classic – a simple free sketching tool with many pre-build components and icons
  https://www.mockplus.com/download/mockplus-classic
- Balsamiq - a rapid low-fidelity UI wireframing tool
  https://balsamiq.com/wireframes/

Automotive User Interfaces

- Course Time Plan
  - Exploration of in-car HMI design via literature study (2 weeks)
  - Generate new ideas based on previous research (1 weeks)
  - Devise initial concept of automotive cockpit (2 weeks)
  - Detailed plan of functional layout (1 week)
  - Implementation and evaluation (4+1 week)

Automotive User Interfaces

- This course will give an overview and an introduction to *practices* and several *libraries* that are commonly used for *creating* UIs in cars
- Main covered topics:
  - Definition of the appearance of the UI of main dashboard
  - Implementation of the self-designed UI in C++ (Qt)

Automotive User Interfaces

- The aim of this course is to describe the in-car user interface design in a manner that minimizes the amount of distraction while maintaining easy access to on-board systems and multimedia services
- You will gain orientation and basic knowledge in the field of interactive technologies connecting the interior of the car with the outside world

Automotive User Interfaces

- Enumerate and characterize ways and means for the interaction of man and the car through the means of a graphical user interface
- Find a sensible way of presenting information to the vehicle crew and suggest ways of its realization
- Identify and assess potentially dangerous ways of notification of the driver while driving the vehicle
- Define the necessary hardware and software resources for the implementation of the designed interface

History of Automotive Infotainment

- 1915 – early mechanical HMIs provided the driver with information such as speed, gas level, or rev counter (almost mechanical HMIs)
- 1922 – the first in-car radio (Ford Model T)
- 1952 – the first in-car phone (more in 70s and 80s)
- 1956 – the first in-car record player
- 1968 – the first in-car cassette player
- 1983 – the first in-car CD player (late 1990s CD-RW, MP3)

History of Automotive Infotainment

- 1990 – the first in-car GPS navigation
- End of the 1990s – mechanical devices were replaced by electronic counterparts. Manufacturers started to aggregate functions within a single device to reduce complexity via one GUI with a hierarchically structured menu
- First in-car infotainment systems combining informative and entertaining functionalities

History of Automotive Infotainment

- General trends – reduce drivers distraction by rearrangement of all instruments (e.g. from footwell to the proximity of the driver) and improvement of comfort
- Nowadays – Internet-based apps, social networks, and extendable, hybrid, adaptive or personalized HMIs are emerging. Motivations for further development are mainly safety, efficiency, and comfort

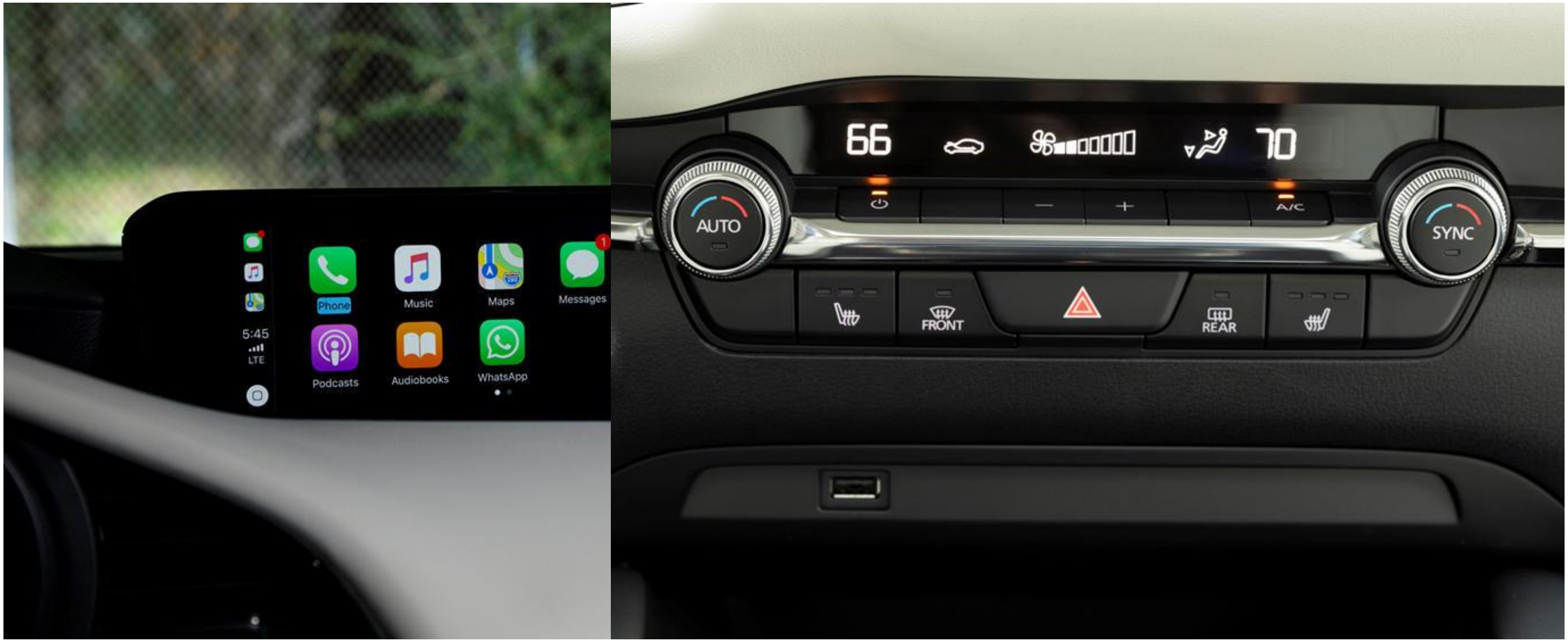
History of Automotive Infotainment

- Example of early infotainment system – car dashboard of Ford Taunus from 1958

Source: Wikimedia Yeti.bigfoot

History of Automotive Infotainment

- Example of contemporary infotainment system – car dashboard of Mazda 3 from 2019



Non-touch screen display and push button assembly *Source: www.carindigo.com*

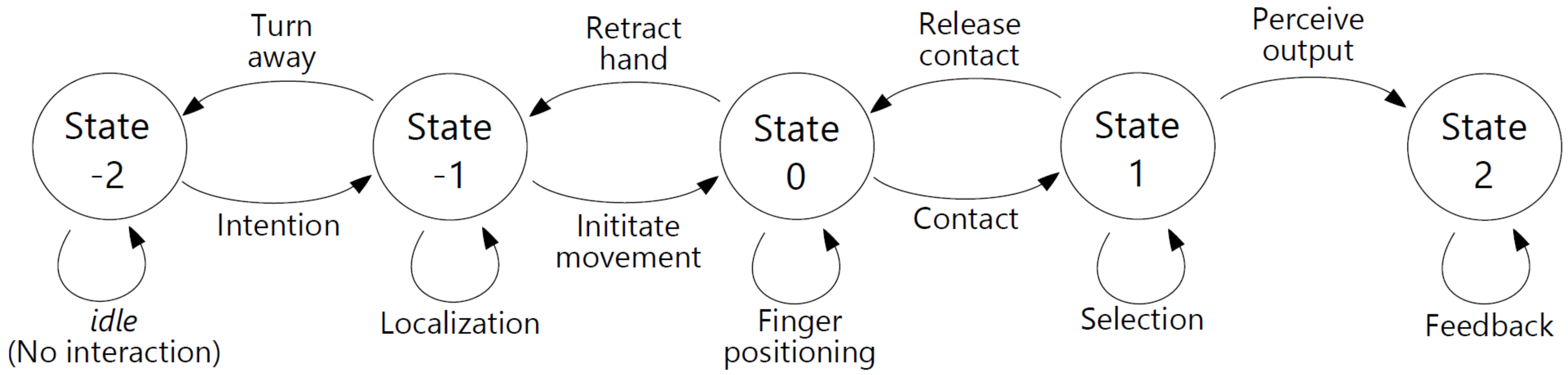
Automotive Infotainment

- Modern HMIs consist of
  - Graphical User Interface (GUI)
  - Speech dialog systems
  - Gesture-based systems (e.g. touch screens, depth/IR sensors)
  - Connectivity to different mobile devices
- The designs from everyday interfaces of the users could be taken as role model for the HMI in the automotive field, e.g. GUI of smartphones

Automotive Infotainment

- But automotive HMIs differ in major points from HMIs in other domains (e.g. attention/distraction - driving must remain the highest priority)
- Transition from manual testing to automated testing methods (scripted test procedures; model-based testing – UML state charts diagram specifying the behavior of HMI; driving simulator/real-world evaluation)

Automotive Infotainment



An example of state model from a user perspective according to related work

Automotive Infotainment

- Electrical interfaces are continually replacing their mechanical counterparts (e.g. mechanical mirror replacement with cameras)
- A big challenge is to design infotainment systems in a way that also people with minor technical background can easily use them
- New tools and methods are necessary to handle the development of more complex functionalities (e.g. UIs based on 3D graphics)

Automotive Infotainment

- Significance of certain development process phases that might have been neglected until now is increasing (e.g. testing of complex HMIs)
- The concept of extendable HMIs (e.g. upgradability of functionality decouples the development cycle from the life cycle of vehicle)

Automotive Infotainment

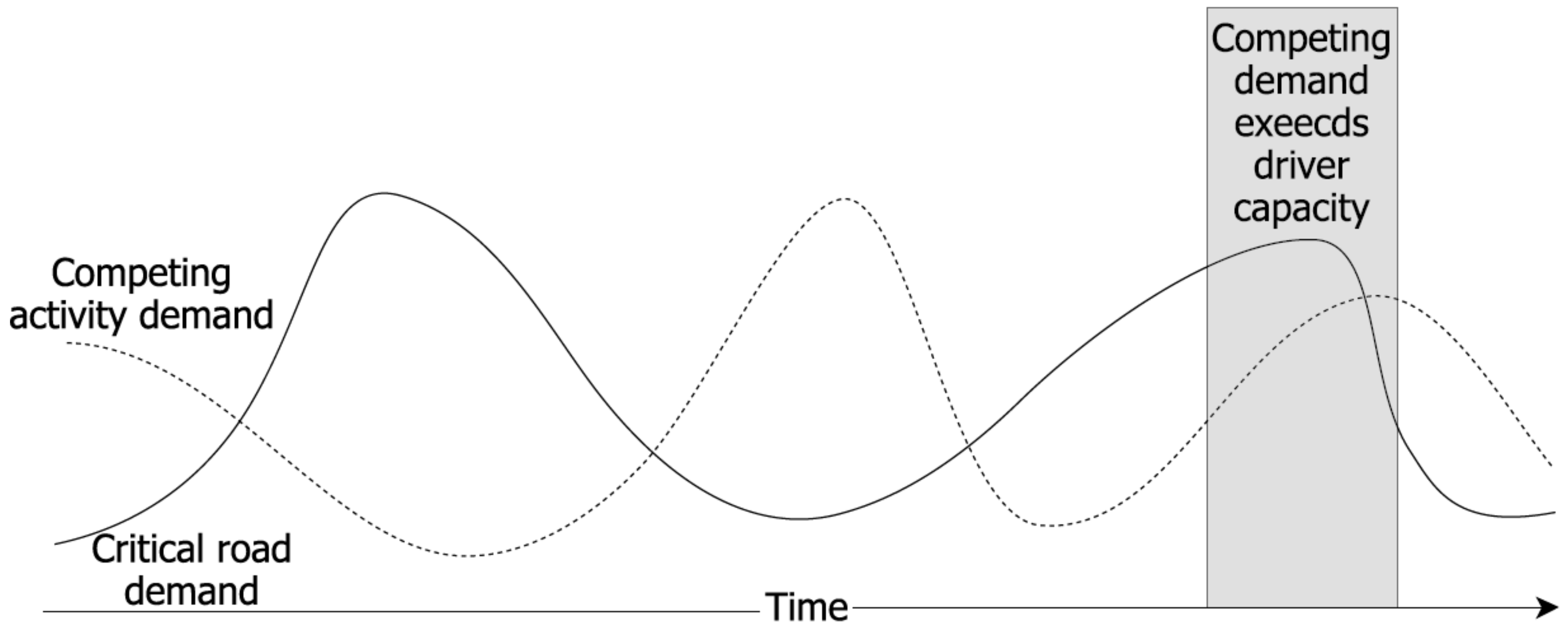
- Applications of in-vehicle infotainment systems include
  - Navigation
  - Media
  - TV
  - Car configuration
  - Data interfaces
  - Telephone
  - Android Auto or Apple CarPlay

Automotive Infotainment

- Reducing distraction
  - Drivers cannot spend their cognitive capacity completely on HCI
  - Cell phone use is estimated to be associated with a minimum of 27% of all accidents
  - Interaction have to be obvious, plausible, and consistent

Automotive Infotainment

- Reducing distraction



Source: Rümelin, S. (2014). *The cockpit for the 21st century*.

Automotive Infotainment

- Categories of distraction
  - Visual distraction – tasks that require the driver to look away from the roadway
  - Manual distraction – tasks that require the driver to take a hand off the steering
  - Cognitive distraction – tasks that required the driver to avert the mental attention

Source: Rümelin, S. (2014). *The cockpit for the 21st century*.

Automotive Infotainment

- Guidelines and requirements for the development of in-vehicle information system (IVIS)
  - ISO 15005 – principles for dialogue management
    - interruptibility of secondary tasks to focus on the driving task
    - one hand always on the steering wheel
    - single information portions should be small and easy to perceive to keep single glances below 1.5 seconds
    - feedback should be given within 250 ms
    - consistent interface design where related functions have a similar presentation (location, orientation, size and coding)

Automotive Infotainment

- Guidelines and requirements for the development of in-vehicle information system (IVIS)
  - ISO 15008 – requirements for visual presentations
    - measurement methods for brightness and contrast
    - minimum dimensions of characters (e.g. a size of 20 arcminutes is acceptable if color is a coding dimension)
    - behavior of dynamically displayed content (e.g. blinking should be avoided except for situations in which immediate attention is required)

Automotive Infotainment

- Guidelines and requirements for the development of in-vehicle information system (IVIS)
  - ISO 3958 – hand-reach envelopes
    - specify the boundaries of locations in which the driver can perform a basic reach task
    - *basic reach task* is defined as controlling a 25 mm control knob with a three-finger grasp without lifting the interacting arm’s shoulder off the seat while the non-reaching hand on the steering wheel and the right foot on the accelerator pedal
    - *extended-finger-operated* and *full-hand-grasped forward control*, extend or reduce the basic envelope by 50 mm

Automotive Infotainment

- Guidelines and requirements for the development of in-vehicle information system (IVIS)
  - European statement of principles on the design of human-machine interface (ESOP)
    - aims at different parties that are involved in the design of in-car systems: vehicle manufacturers; after-market system and service producers; manufacturers of parts enabling the use of nomadic devices by the driver while driving; service providers including software providers or broadcasters of information meant to be used by the driver

Automotive Infotainment

- Other demands
  - The system should be easy to understand and not distract, but enable the driver to choose whether, when and how an interaction takes place
  - The driver should always be able to keep at least one hand on the steering wheel while interacting with the system
  - Visual displays should be positioned as close as practicable to the driver’s normal line of sight
  - Displays that contain relevant information and where long glance sequences are expected, should not be placed below approximately 30° downward the viewing angle of the driver’s normal forward view

Automotive Infotainment

- Other recommendations (by NHTSA in USA)
  - Not to include video and automatically scrolling text
  - Not to allow manual text entry to perform text-based communication or internet browsing
  - Displaying any text and graphical or photographic images should be avoided
  - Interaction should not require glances longer than 2 seconds
  - Cumulated glance durations for should not exceed 12 seconds

Note that implementing these guidelines would mean to exclude most of the functionality available in current in-car systems, at least for the control during driving.

Input/Output Devices

- Physical components (in limited space of a car)
  - Push button assembly – enables direct access for major infotainment functions or functions that are frequently used (ESC, door lock, voice control)
  - Some buttons are duplicated on the steering wheel
  - Such button may use indicator lights or miniaturized displays
  - „Buttons“ may be also realized as capacitive surfaces or proximity sensors

Input/Output Devices

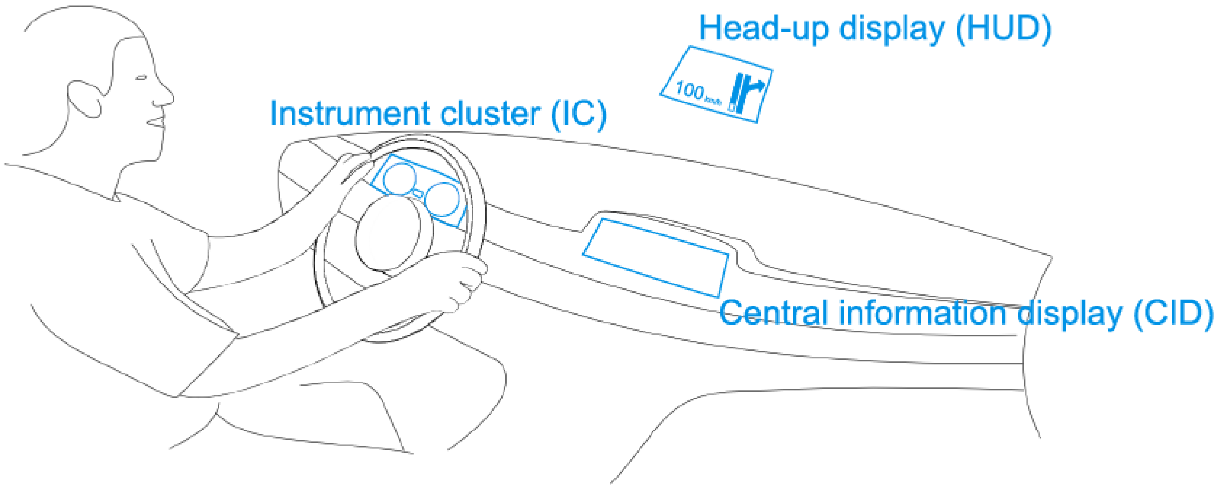
- Physical components (in limited space of a car)
  - Center control elements (CCEs) – multi-purpose rotary controllers with force feedback surrounded by buttons for switching between the contexts or providing quick access to common functions, and sometimes even with touchpads (MMI in Audi, iDrive in BMW, Command in Mercedes-Benz)

Input/Output Devices

- Physical components (in limited space of a car)
  - Voice control – enables the user to input command in natural language without using hands using the technique of Hidden Markov Models for estimating the most probable word sequence
  - (Non-)touchscreen central displays – enabling direct manipulation of interactive objects by means of touch and gestures when equipped with resistive or capacitive surfaces

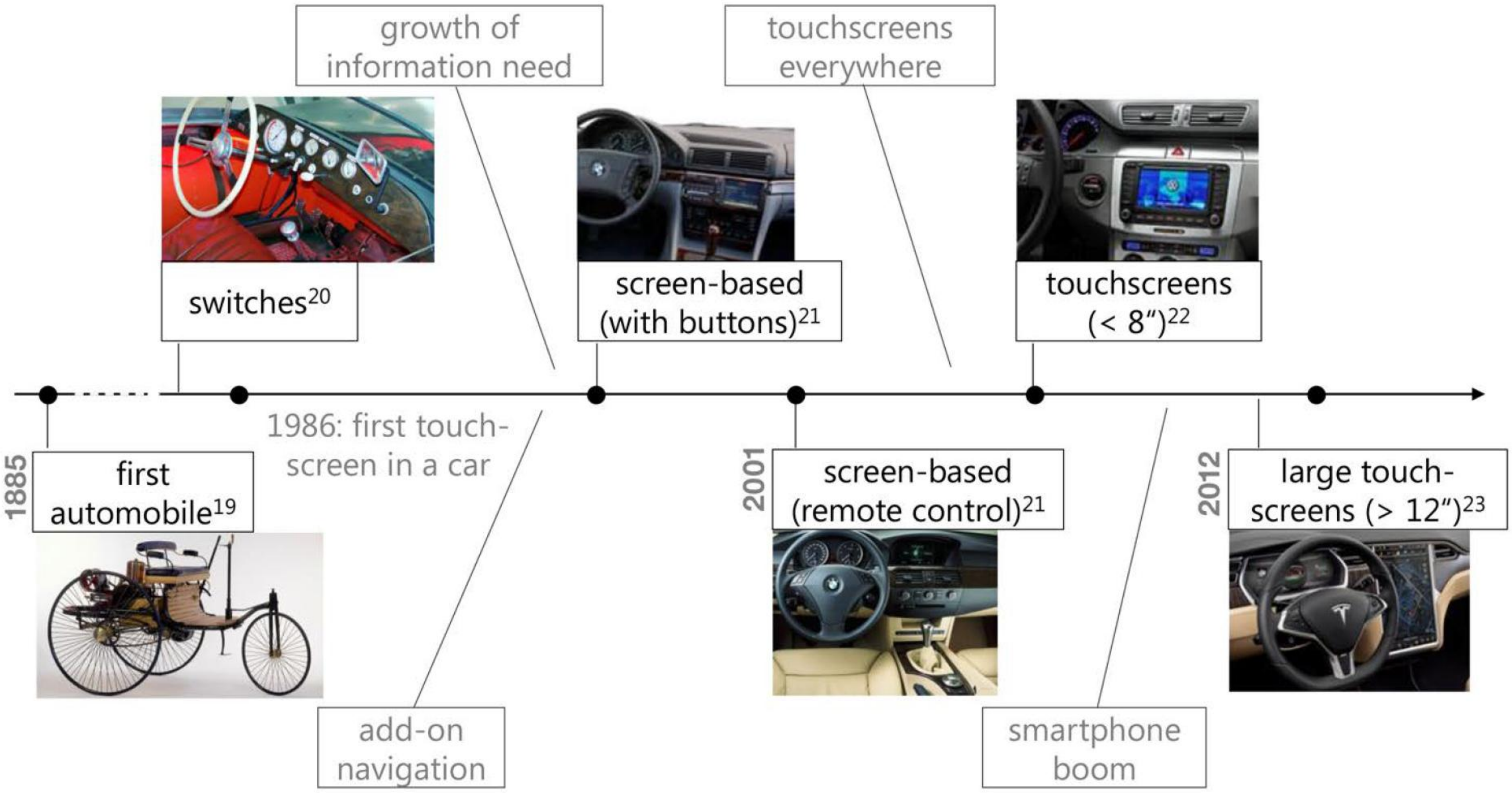
Input/Output Devices

- Physical components (in limited space of a car)
  - Head-down displays (HDDs) – central information display (CID) or instrumental cluster (IC) – the road scene vanishes from the line of sight
  - Automotive head-up display (HUD) or head-mounted display (HMD) present the content as an overlay to the road e line of sight



Source: Rümelin, S. (2014). *The cockpit for the 21st century*.

Evolution of Infotainment Systems

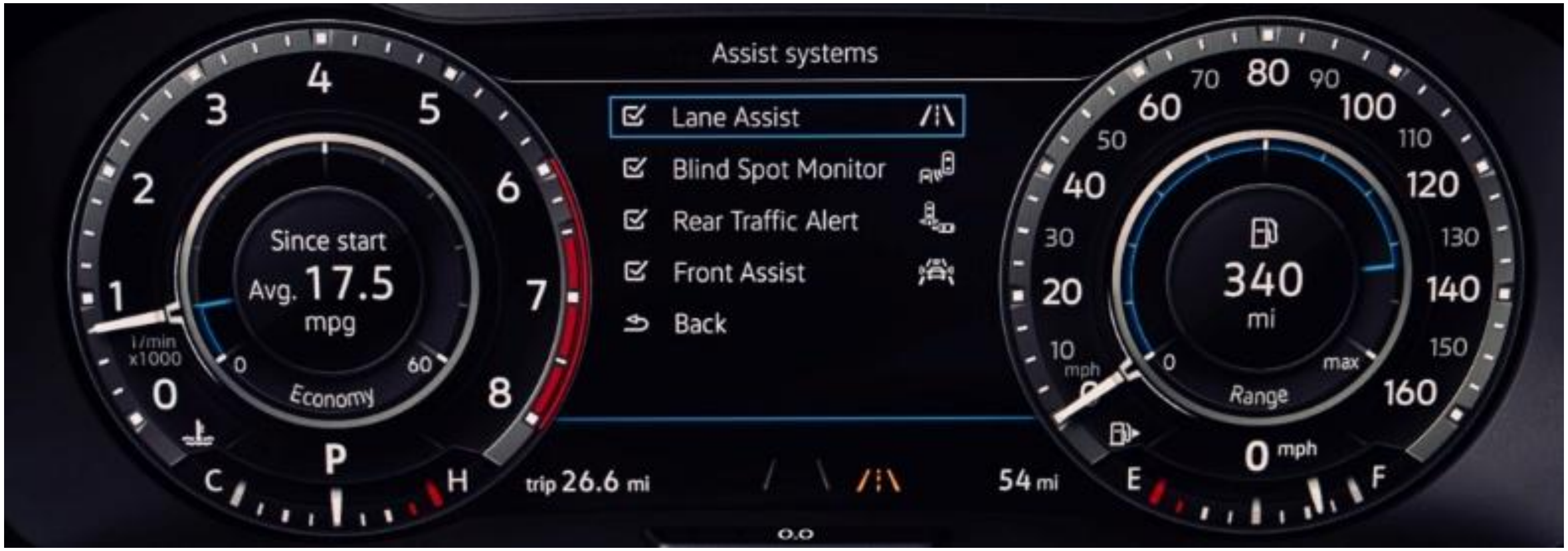

growth of information need
touchscreens everywhere
switches[20]
screen-based (with buttons)[21]
touchscreens (< 8″)[22]
1986: first touch-screen in a car
1885
first automobile[19]
2001
screen-based (remote control)[21]
2012
large touch-screens (> 12″)[23]
add-on navigation
smartphone boom

Source: Rümelin, S. (2014). *The cockpit for the 21st century*.

Input/Output Devices

VW Touareg Innovision Cockpit with high-res displays placed in the head unit and the instrument cluster replacing the classic elements in front of the driver

Current Infotainment Systems



Volkswagen Digital Cockpit

Designing User Interfaces

- UI is one of the most important component of a computer-based system
- Poorly designed UI makes it difficult to use the potential of the implemented functionality
- Three basic principles of effective UI design
  - put the user in control
  - reduce the user's memory load
  - make the UI consistent

Mandel‘s Rules

- Define interaction modes in a way that does not force a user into **unnecessary or undesired actions**.
- Provide for **flexible interaction** - software might allow a user to interact via keyboard commands, mouse movement, a digitizer pen, or voice recognition commands
- Allow user interaction to be **interruptible and undoable**.
- Streamline interaction as skill levels advance and allow the interaction to be customized - repeated execution of the same sequence of interactions can be replaced by a macro mechanism.
- **Hide technical internals** from the casual user.

Source: Mandel, T., The Elements of User Interface Design, Wiley, 1997.

Mandel‘s Rules

- Design for **direct interaction** with objects that appear on the screen - perform the task in a similar manner as if it were a physical thing.
- Reduce the user’s memory load using the following hints…
- **Reduce demand on short-term memory** - reduce the requirement to remember past actions and results by providing visual cues that enable a user to recognize past actions
- Establish **meaningful defaults** - the initial set of defaults should make sense for the average user and allow reset functionality

Source: Mandel, T., The Elements of User Interface Design, Wiley, 1997.

Mandel‘s Rules

- **Define shortcuts** that are intuitive - the mnemonic should be tied to the action in a way that is easy to remember
- The visual layout of the interface should be based on a **real world metaphor**.
- **Disclose information in a progressive fashion** - the interface should be organized hierarchically, e.g., more detail should be presented after the user indicates interest with a mouse pick.

Source: Mandel, T., The Elements of User Interface Design, Wiley, 1997.

Designing User Interfaces

- General 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 Interfaces

- User Analysis (relation between the user and the product)
    - 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 Interfaces

- 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 Interfaces

- 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 Interfaces

- 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

Application Design

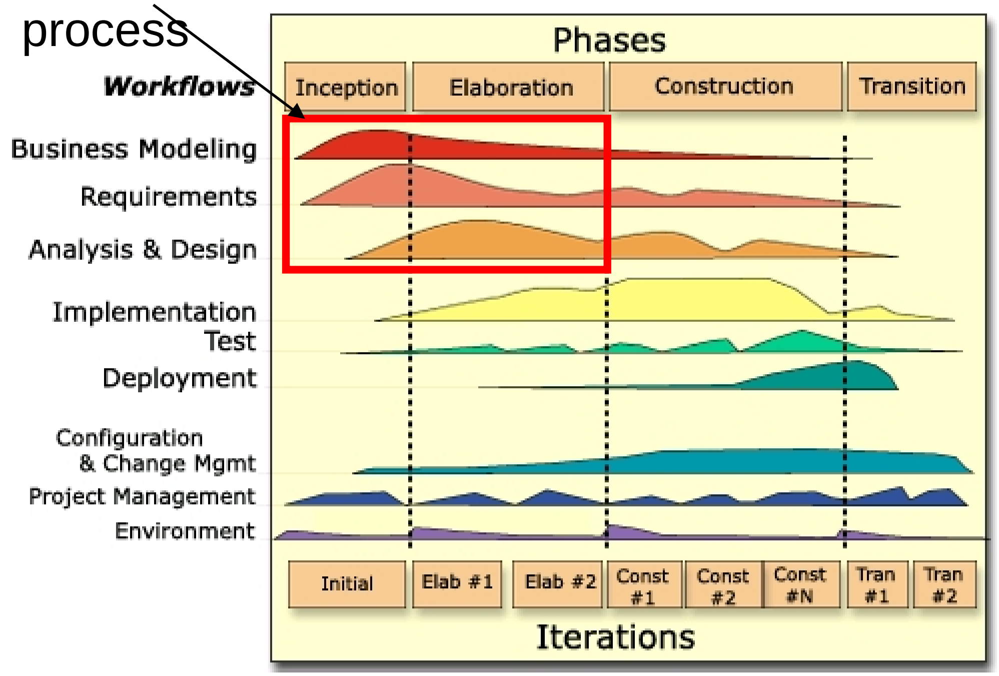
- What should influence the application design:
  - User skills, workflow, habits, and expectations
  - User should be involved in the design process

  - 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

- RUP Process Architecture

UI design process



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

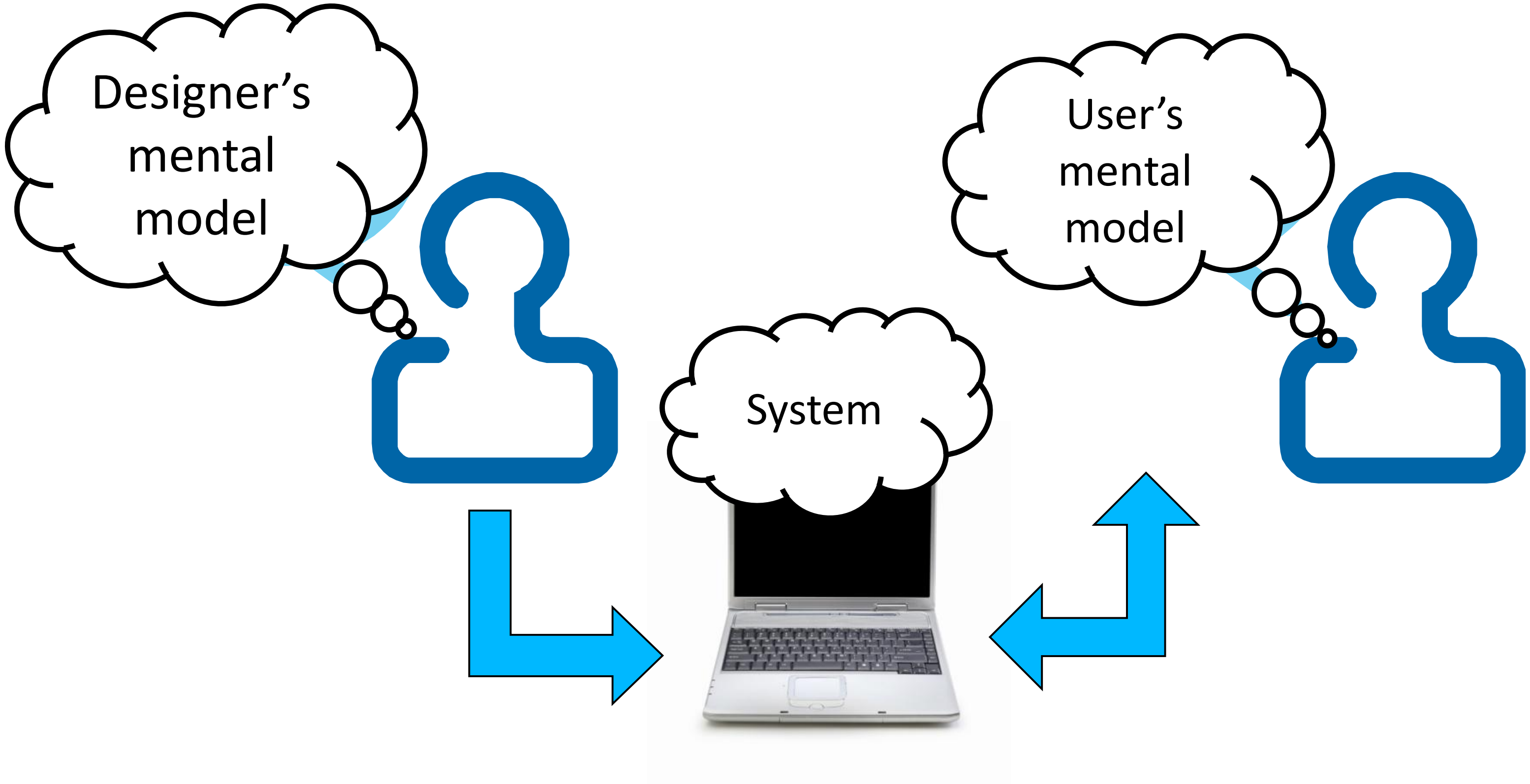
Mental model

- 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 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

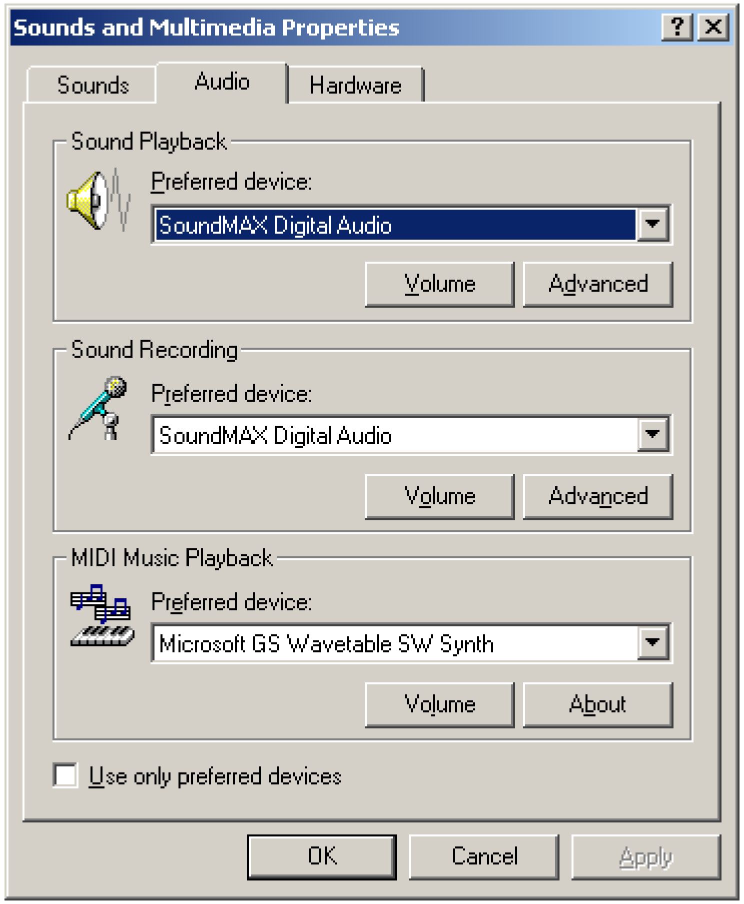


- Worst-case scenario: Developers often have a flawed mental model of their own software and a real user's mental model is quite different
- Expression of mental models: Flow diagrams – a way to express a dynamic systems

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 they will experience frustration, dubiousness, etc.
- We need to choose interaction patterns that fit the mental model of the user
- The technical realization is responsible for the adequate implementation of the underlying concept

Mental model

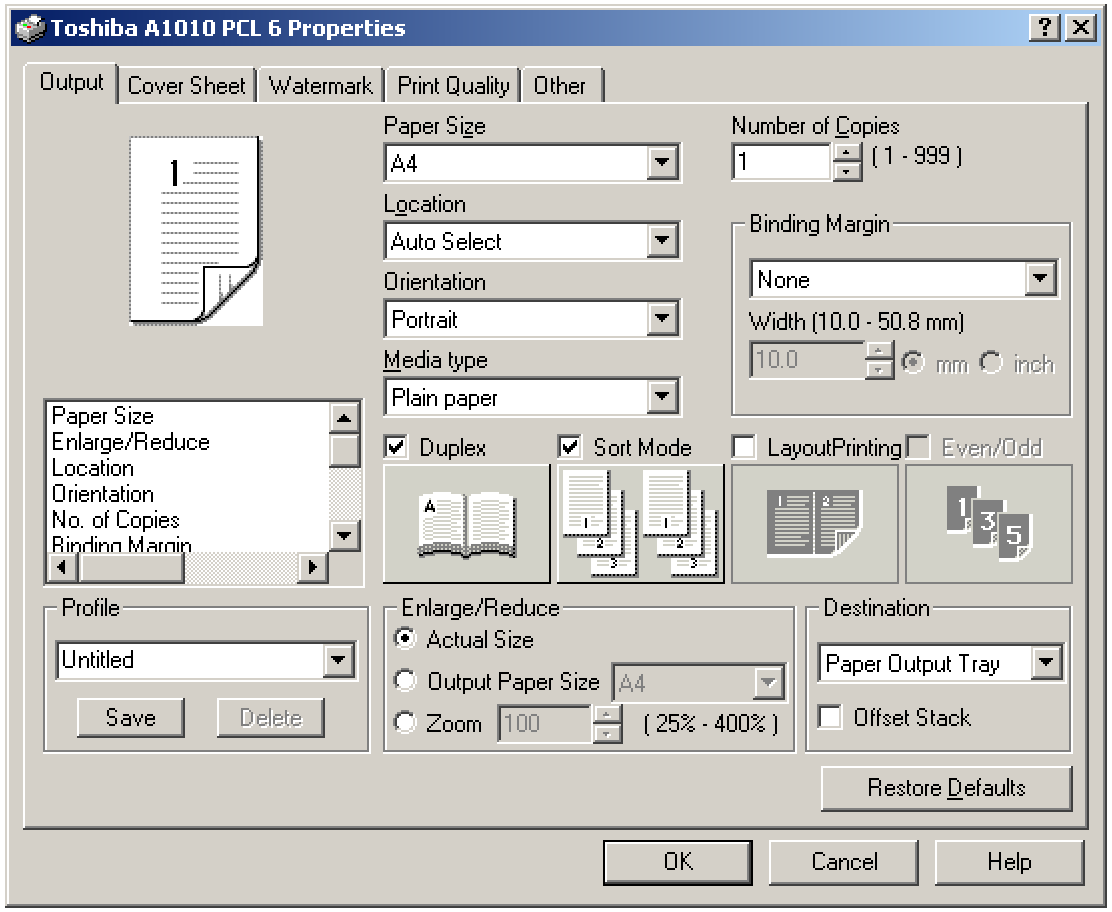


Sound Playback
Preferred device: SoundMAX Digital Audio

Sound Recording
Preferred device: SoundMAX Digital Audio

MIDI Music Playback
Preferred device: Microsoft GS Wavetable SW Synth

☐ Use only preferred devices



Paper Size: A4
Location: Auto Select
Orientation: Portrait
Media type: Plain paper
Number of Copies: 1 (1 - 999)

Binding Margin: None
Width (10.0 - 50.8 mm): 10.0 ● mm ○ inch

☑ Duplex ☑ Sort Mode ☐ LayoutPrinting ☐ Even/Odd

Profile: Untitled

Enlarge/Reduce
● Actual Size
○ Output Paper Size A4
○ Zoom 100 (25% - 400%)

Destination: Paper Output Tray
☐ Offset Stack

Which user interface is easier to grasp?

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/

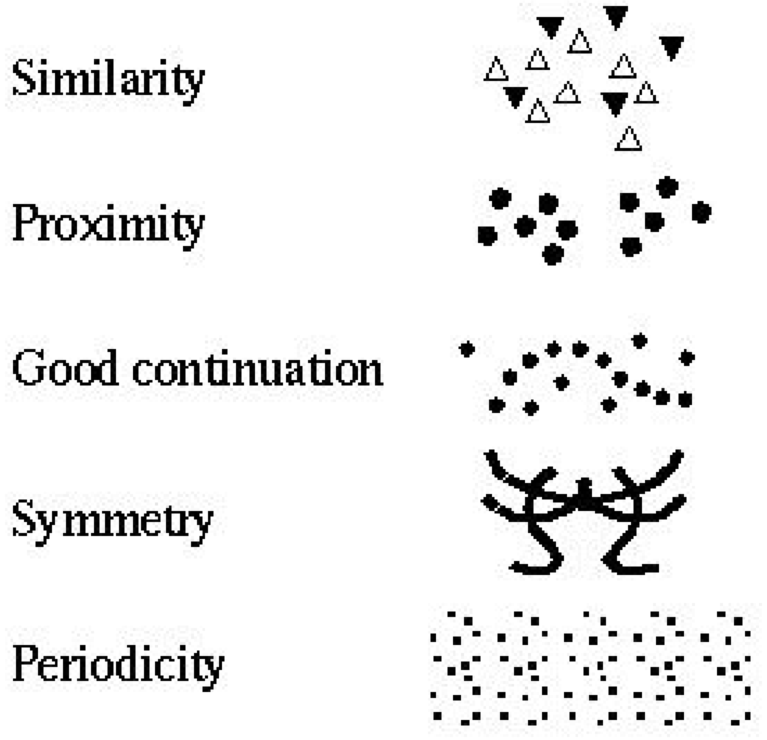
Gestalt theory

“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

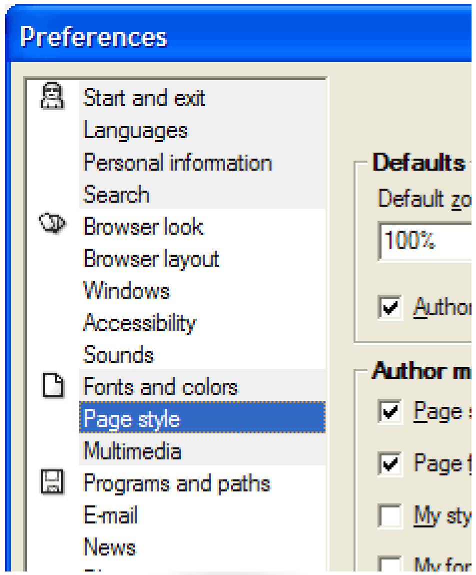
Gestalt theory

- Provides common organization principles used regularly in visual design



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**

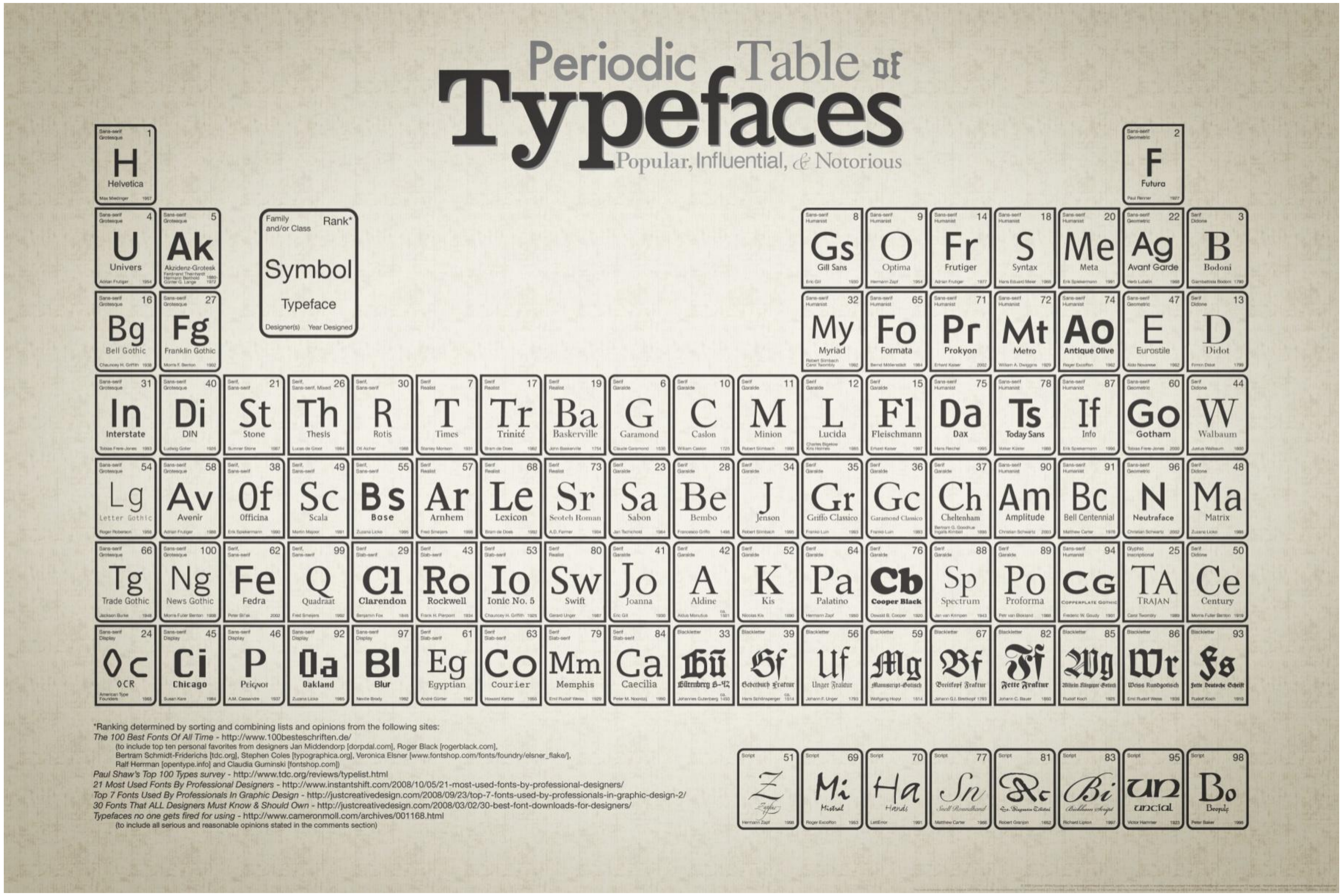
Full Control

- **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

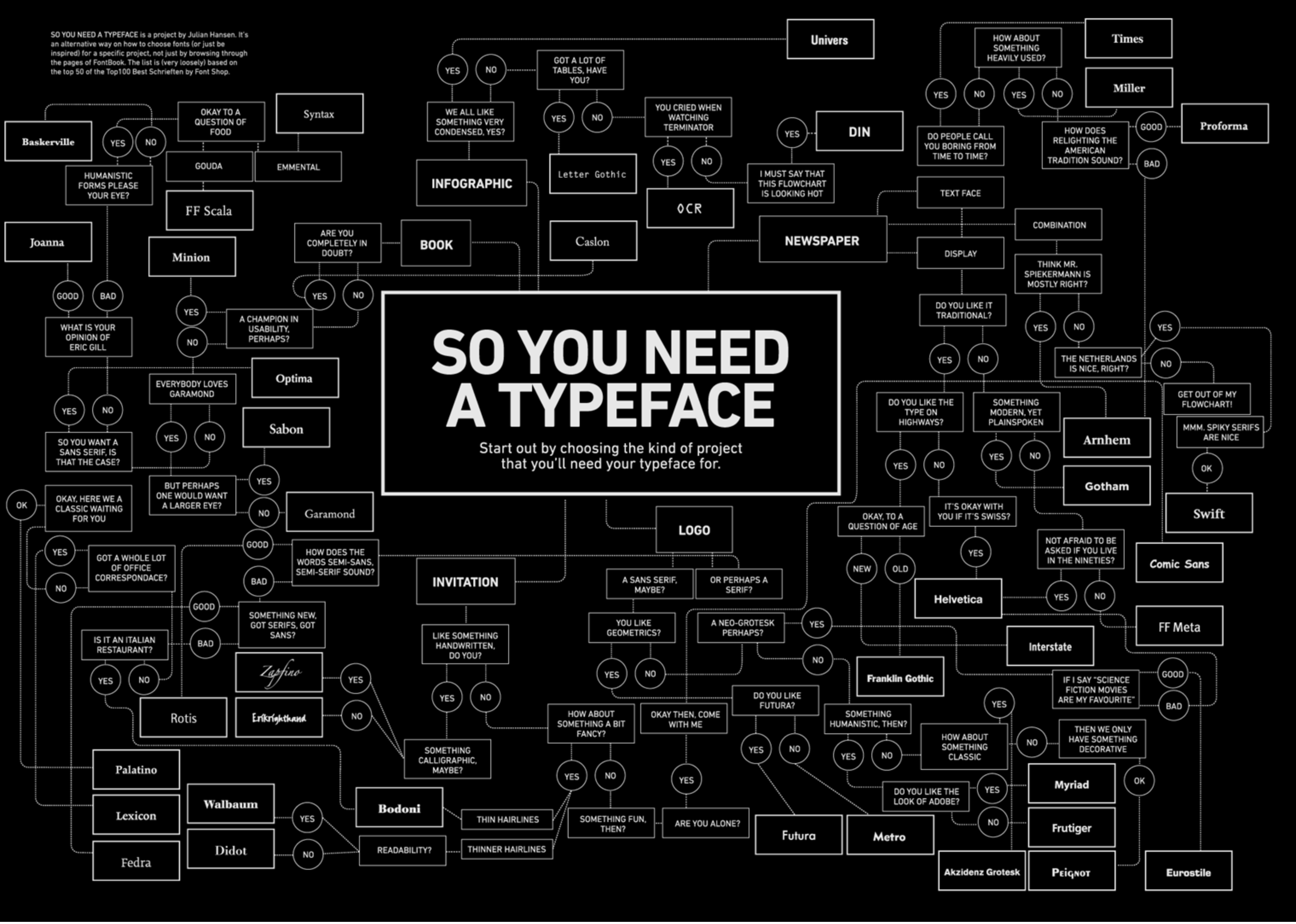
- People have a **limited short-term memory**
- Having to keep track of several things at once can leave a user frustrated or incapable of using your interface
- Try and consolidate multiple pages, reduce unneeded motion, and generally **just keep things simple**

Typefaces



*Ranking determined by sorting and combining lists and opinions from the following sites:
The 100 Best Fonts Of All Time - http://www.100besteschriften.de/
(to include top ten personal favorites from designers Jan Middendorp [dorpdal.com], Roger Black [rogerblack.com], Bertram Schmidt-Friderichs [tdc.org], Stephen Coles [typographica.org], Veronica Elsner [www.fontshop.com/fonts/foundry/elsner_flake/], Ralf Herrman [opentype.info] and Claudia Guminski [fontshop.com])
Paul Shaw's Top 100 Types survey - http://www.tdc.org/reviews/typelist.html
21 Most Used Fonts By Professional Designers - http://www.instantshift.com/2008/10/05/21-most-used-fonts-by-professional-designers/
Top 7 Fonts Used By Professionals In Graphic Design - http://justcreativedesign.com/2008/09/23/top-7-fonts-used-by-professionals-in-graphic-design-2/
30 Fonts That ALL Designers Must Know & Should Own - http://justcreativedesign.com/2008/03/02/30-best-font-downloads-for-designers/
Typefaces no one gets fired for using - http://www.cameronmoll.com/archives/001168.html
(to include all serious and reasonable opinions stated in the comments section)

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 Schriften by Font Shop.

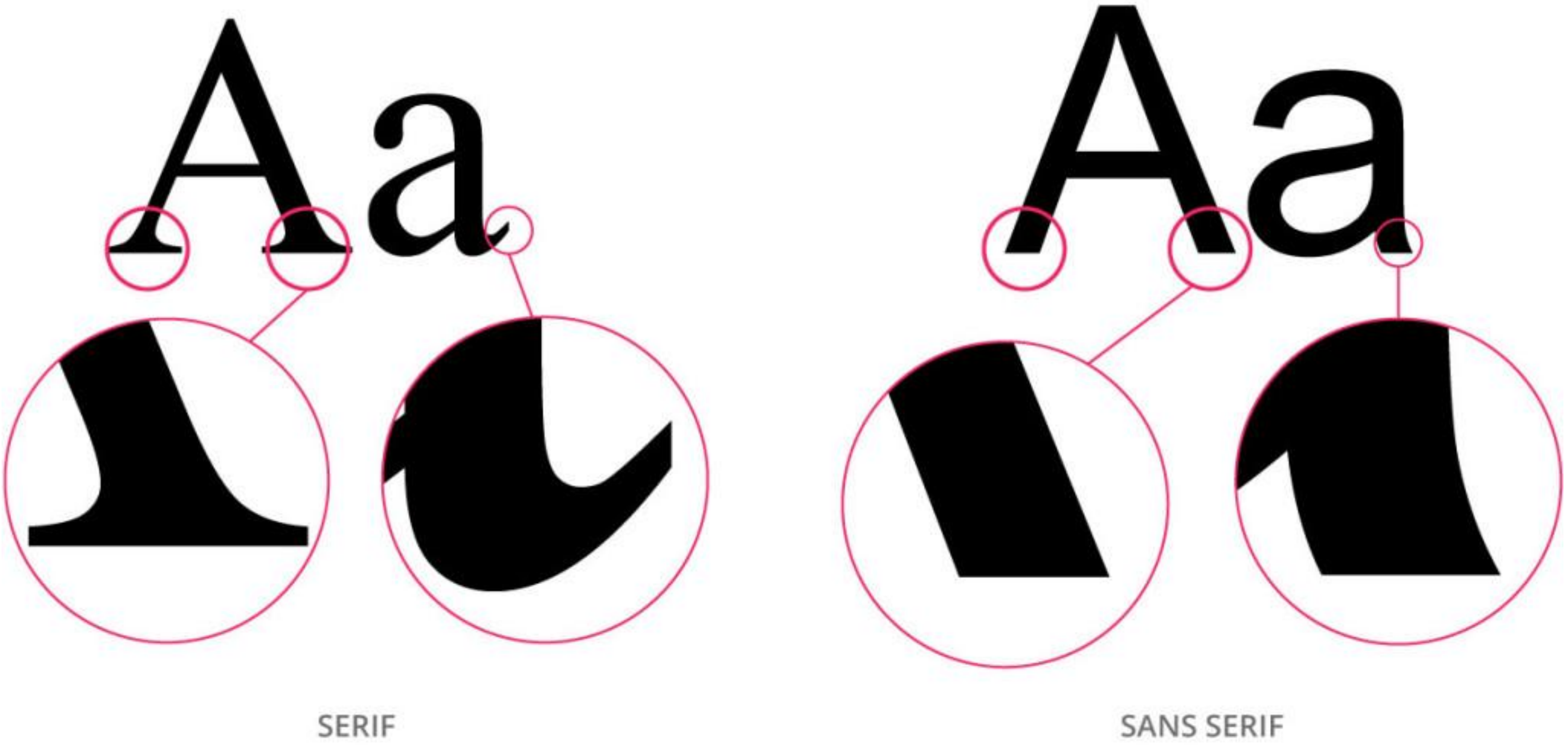
SO YOU NEED A TYPEFACE
Start out by choosing the kind of project that you'll need your typeface for.
BOOK
NEWSPAPER
INFOGRAPHIC
LOGO
INVITATION
Univers
GOT A LOT OF TABLES, HAVE YOU?
WE ALL LIKE SOMETHING VERY CONDENSED, YES?
Letter Gothic
YOU CRIED WHEN WATCHING TERMINATOR
OCR
DIN
I MUST SAY THAT THIS FLOWCHART IS LOOKING HOT
Caslon
ARE YOU COMPLETELY IN DOUBT?
A CHAMPION IN USABILITY, PERHAPS?
Minion
Optima
EVERYBODY LOVES GARAMOND
Sabon
BUT PERHAPS ONE WOULD WANT A LARGER EYE?
Garamond
HOW DOES THE WORDS SEMI-SANS, SEMI-SERIF SOUND?
SOMETHING NEW, GOT SERIFS, GOT SANS?
Rotis
Baskerville
OKAY TO A QUESTION OF FOOD
Syntax
GOUDA
EMMENTAL
FF Scala
HUMANISTIC FORMS PLEASE YOUR EYE?
Joanna
WHAT IS YOUR OPINION OF ERIC GILL
SO YOU WANT A SANS SERIF, IS THAT THE CASE?
OKAY, HERE WE A CLASSIC WAITING FOR YOU
GOT A WHOLE LOT OF OFFICE CORRESPONDACE?
IS IT AN ITALIAN RESTAURANT?
Palatino
Lexicon
Fedra
Walbaum
Didot
READABILITY?
Bodoni
THIN HAIRLINES
THINNER HAIRLINES
Zapfino
Erikrighthand
SOMETHING CALLIGRAPHIC, MAYBE?
LIKE SOMETHING HANDWRITTEN, DO YOU?
HOW ABOUT SOMETHING A BIT FANCY?
SOMETHING FUN, THEN?
ARE YOU ALONE?
OKAY THEN, COME WITH ME
A SANS SERIF, MAYBE?
OR PERHAPS A SERIF?
YOU LIKE GEOMETRICS?
A NEO-GROTESK PERHAPS?
DO YOU LIKE FUTURA?
Futura
Metro
Franklin Gothic
SOMETHING HUMANISTIC, THEN?
HOW ABOUT SOMETHING CLASSIC
DO YOU LIKE THE LOOK OF ADOBE?
Myriad
Frutiger
Akzidenz Grotesk
Peignot
THEN WE ONLY HAVE SOMETHING DECORATIVE
Eurostile
IF I SAY "SCIENCE FICTION MOVIES ARE MY FAVOURITE"
Interstate
FF Meta
NOT AFRAID TO BE ASKED IF YOU LIVE IN THE NINETIES?
Helvetica
IT'S OKAY WITH YOU IF IT'S SWISS?
OKAY, TO A QUESTION OF AGE
NEW
OLD
DO YOU LIKE THE TYPE ON HIGHWAYS?
SOMETHING MODERN, YET PLAINSPOKEN
Gotham
Arnhem
DO YOU LIKE IT TRADITIONAL?
TEXT FACE
COMBINATION
DISPLAY
THINK MR. SPIEKERMANN IS MOSTLY RIGHT?
THE NETHERLANDS IS NICE, RIGHT?
GET OUT OF MY FLOWCHART!
MMM. SPIKY SERIFS ARE NICE
Swift
Comic Sans
DO PEOPLE CALL YOU BORING FROM TIME TO TIME?
HOW ABOUT SOMETHING HEAVILY USED?
Times
Miller
HOW DOES RELIGHTING THE AMERICAN TRADITION SOUND?
Proforma
YES
NO
GOOD
BAD
OK

Source: http://www.julianhansen.com/#/zimmer/

Typefaces vs. Fonts

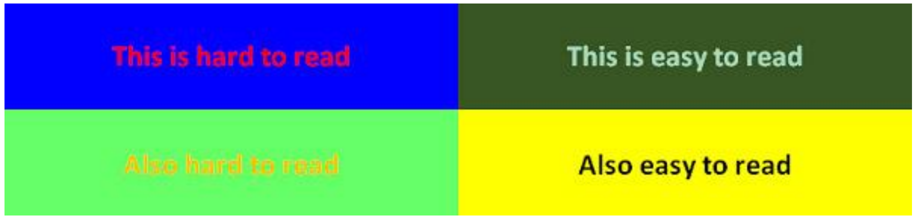
- **Typography** – the art and technique that consists of arranging type (form) with the purpose of writing
  - The purpose is to make sure the text is easy to read
- **Typeface** – set of typographical symbols and characters (letters, numbers, and other chars)
  - Verdana
- **Type family** – group of 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



Basic Principles of Typography

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



This is hard to read	This is easy to read
Also hard to read	Also easy to read

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	Script
ABCabc	ABCabc
(Vineta BT)	(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 handwritten characters and symbols

Typefaces

Dingbat

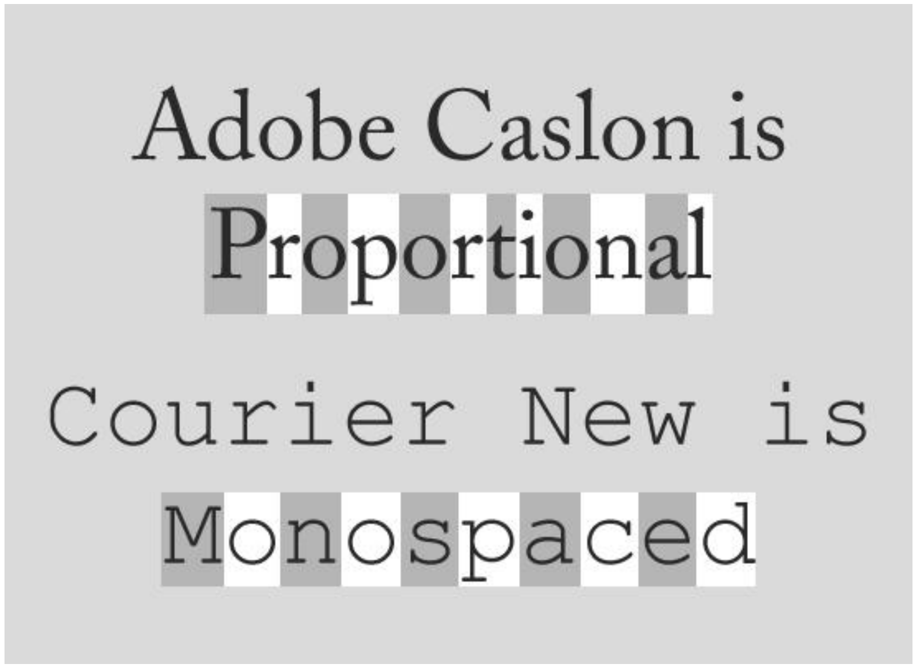
♥∂∑ℜ∅

(Symbol)

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

Typefaces

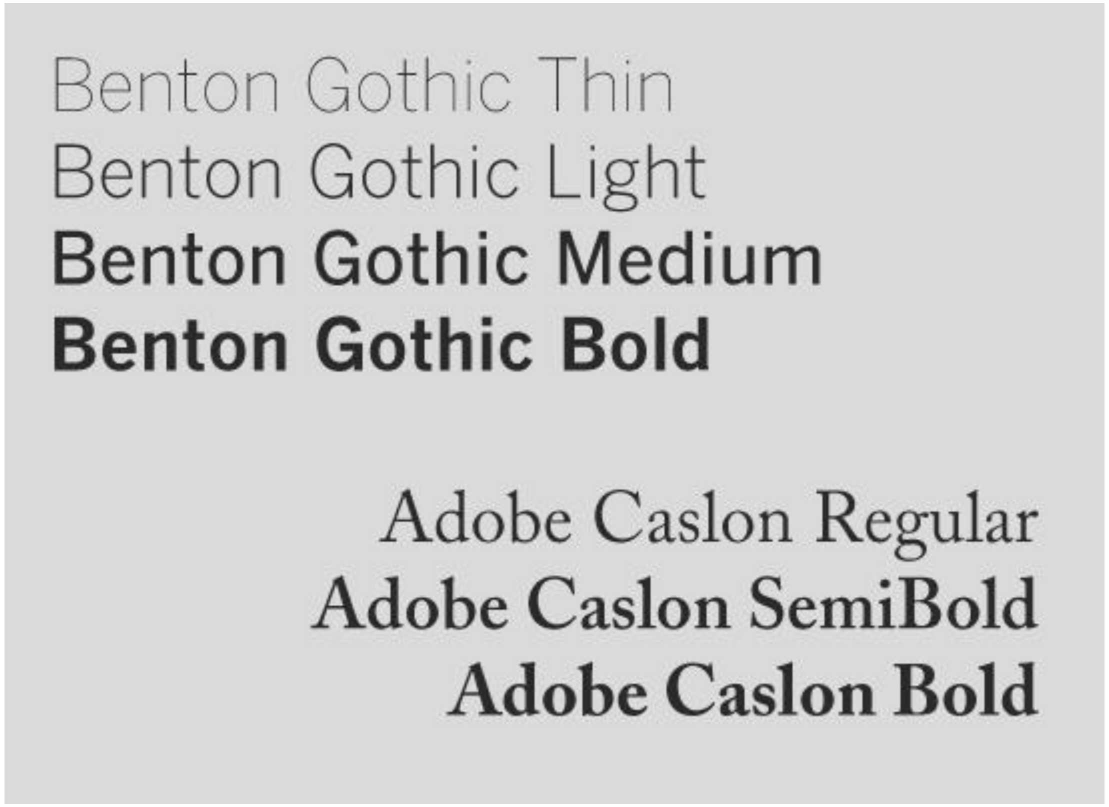
- Proportional – the space a character takes up is dependent on the natural width of that character
- Monospaced – each character takes up the same amount of space



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

Typefaces

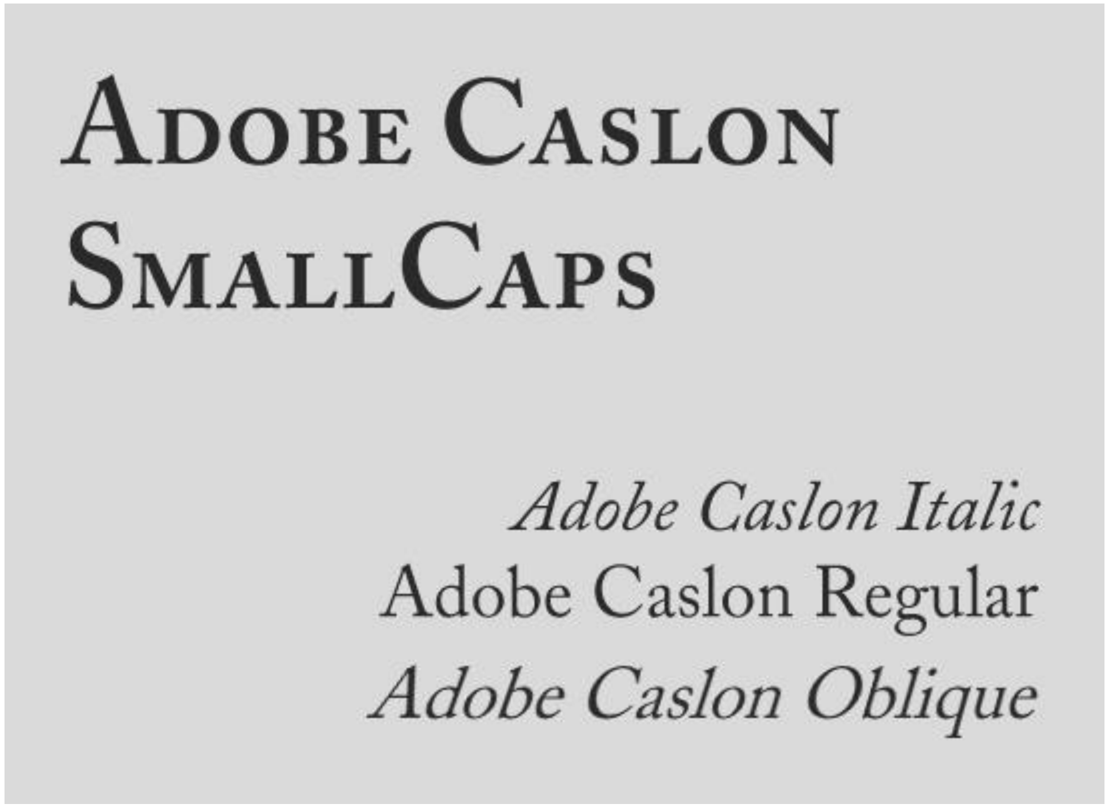
- Weight – refers to the thickness of the strokes that make up the characters



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

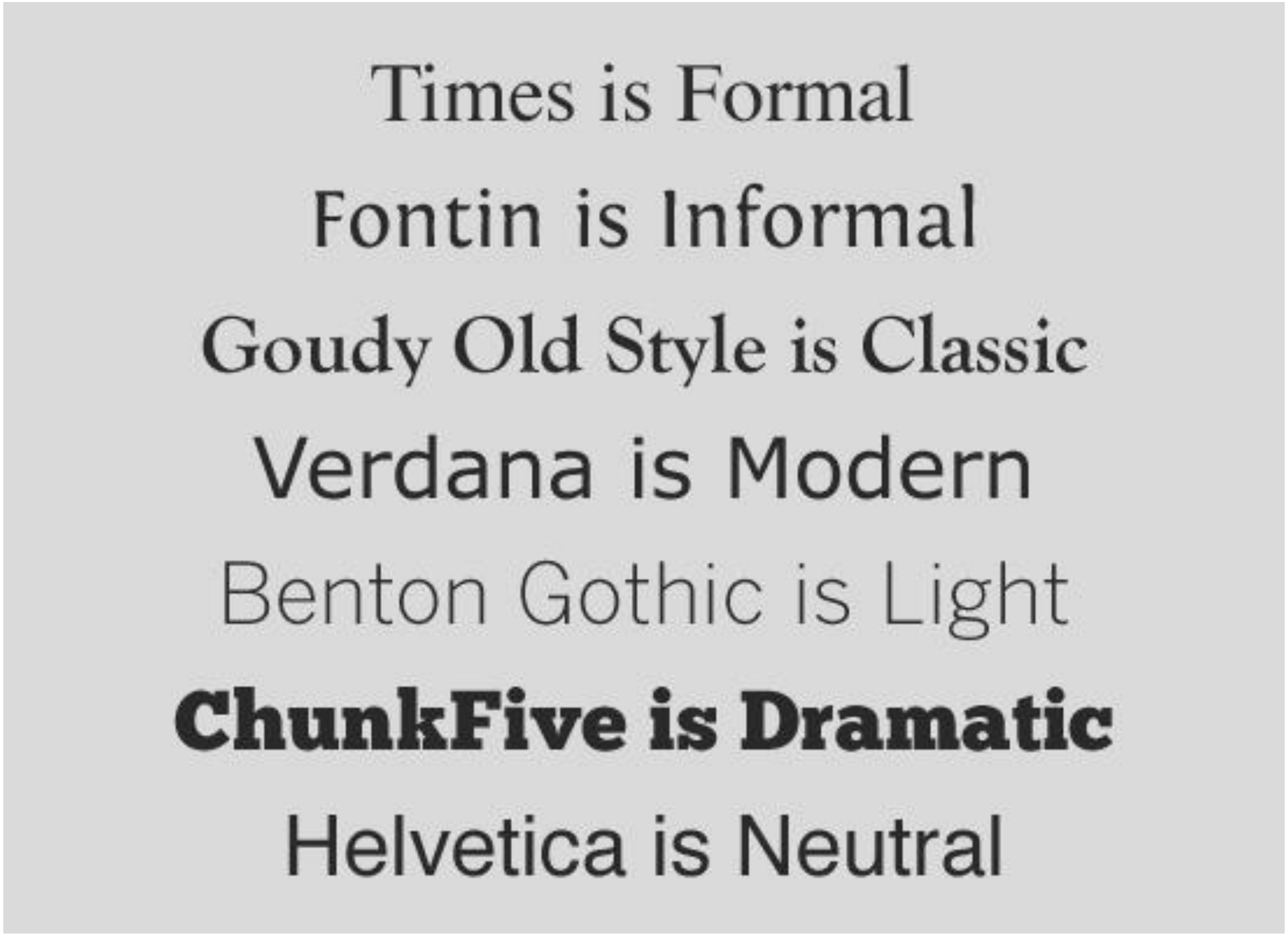
Typefaces

- 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



Times is Formal

Fontin is Informal

Goudy Old Style is Classic

Verdana is Modern

Benton Gothic is Light

ChunkFive is Dramatic

Helvetica is Neutral

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

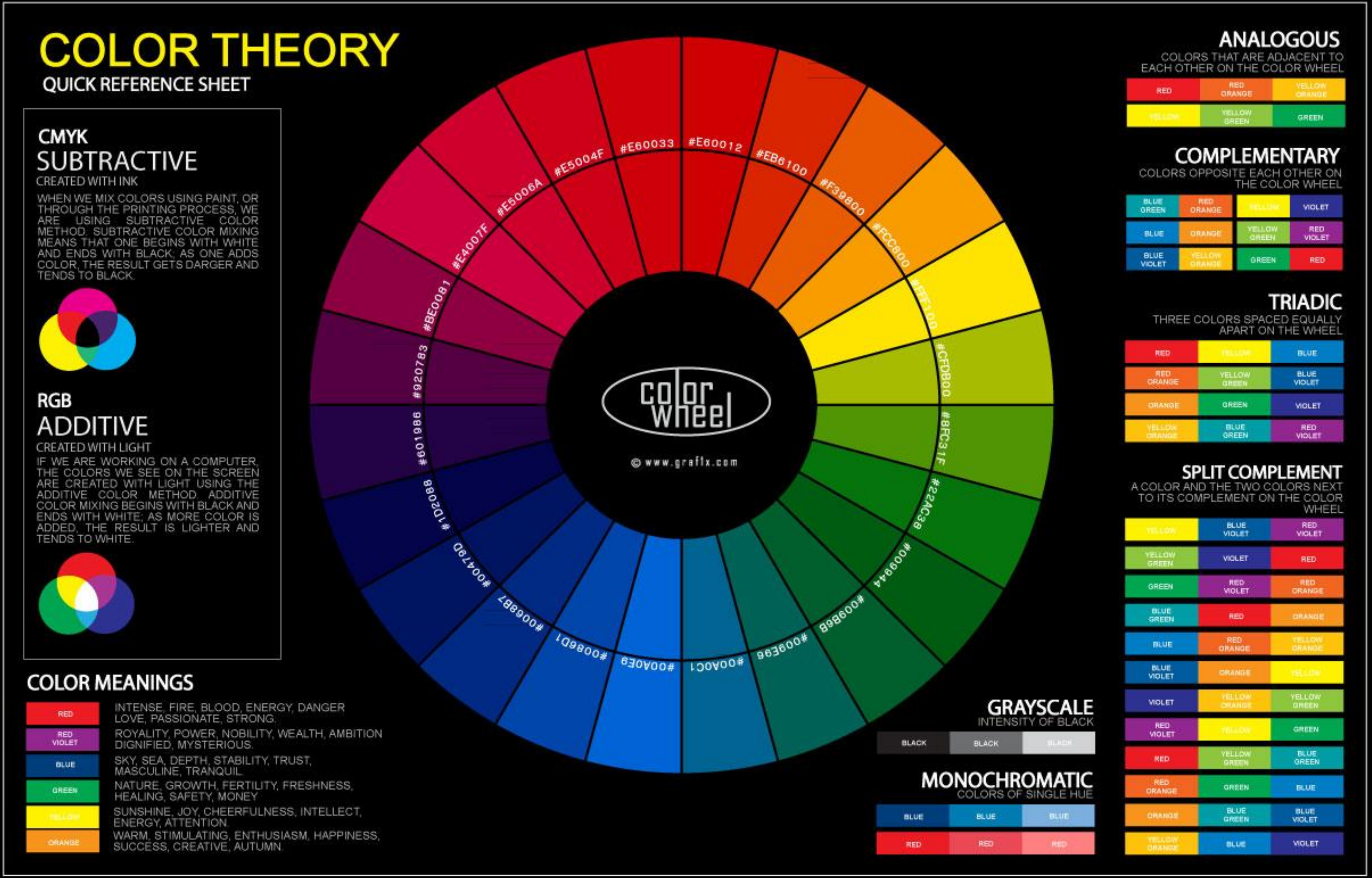
Which Font?

- 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 THEORY

QUICK REFERENCE SHEET

CMYK SUBTRACTIVE

CREATED WITH INK

WHEN WE MIX COLORS USING PAINT, OR THROUGH THE PRINTING PROCESS, WE ARE USING SUBTRACTIVE COLOR METHOD. SUBTRACTIVE COLOR MIXING MEANS THAT ONE BEGINS WITH WHITE AND ENDS WITH BLACK; AS ONE ADDS COLOR, THE RESULT GETS DARGER AND TENDS TO BLACK.

RGB ADDITIVE

CREATED WITH LIGHT

IF WE ARE WORKING ON A COMPUTER, THE COLORS WE SEE ON THE SCREEN ARE CREATED WITH LIGHT USING THE ADDITIVE COLOR METHOD. ADDITIVE COLOR MIXING BEGINS WITH BLACK AND ENDS WITH WHITE; AS MORE COLOR IS ADDED, THE RESULT IS LIGHTER AND TENDS TO WHITE.

COLOR MEANINGS

RED	INTENSE, FIRE, BLOOD, ENERGY, DANGER LOVE, PASSIONATE, STRONG.
RED VIOLET	ROYALITY, POWER, NOBILITY, WEALTH, AMBITION DIGNIFIED, MYSTERIOUS.
BLUE	SKY, SEA, DEPTH, STABILITY, TRUST, MASCULINE, TRANQUIL.
GREEN	NATURE, GROWTH, FERTILITY, FRESHNESS, HEALING, SAFETY, MONEY
YELLOW	SUNSHINE, JOY, CHEERFULNESS, INTELLECT, ENERGY, ATTENTION.
ORANGE	WARM, STIMULATING, ENTHUSIASM, HAPPINESS, SUCCESS, CREATIVE, AUTUMN.

GRAYSCALE

INTENSITY OF BLACK

BLACK	BLACK	BLACK

MONOCHROMATIC

COLORS OF SINGLE HUE

BLUE	BLUE	BLUE
RED	RED	RED

ANALOGOUS

COLORS THAT ARE ADJACENT TO EACH OTHER ON THE COLOR WHEEL

RED	RED ORANGE	YELLOW ORANGE
YELLOW	YELLOW GREEN	GREEN

COMPLEMENTARY

COLORS OPPOSITE EACH OTHER ON THE COLOR WHEEL

BLUE GREEN	RED ORANGE	YELLOW	VIOLET
BLUE	ORANGE	YELLOW GREEN	RED VIOLET
BLUE VIOLET	YELLOW ORANGE	GREEN	RED

TRIADIC

THREE COLORS SPACED EQUALLY APART ON THE WHEEL

RED	YELLOW	BLUE
RED ORANGE	YELLOW GREEN	BLUE VIOLET
ORANGE	GREEN	VIOLET
YELLOW ORANGE	BLUE GREEN	RED VIOLET

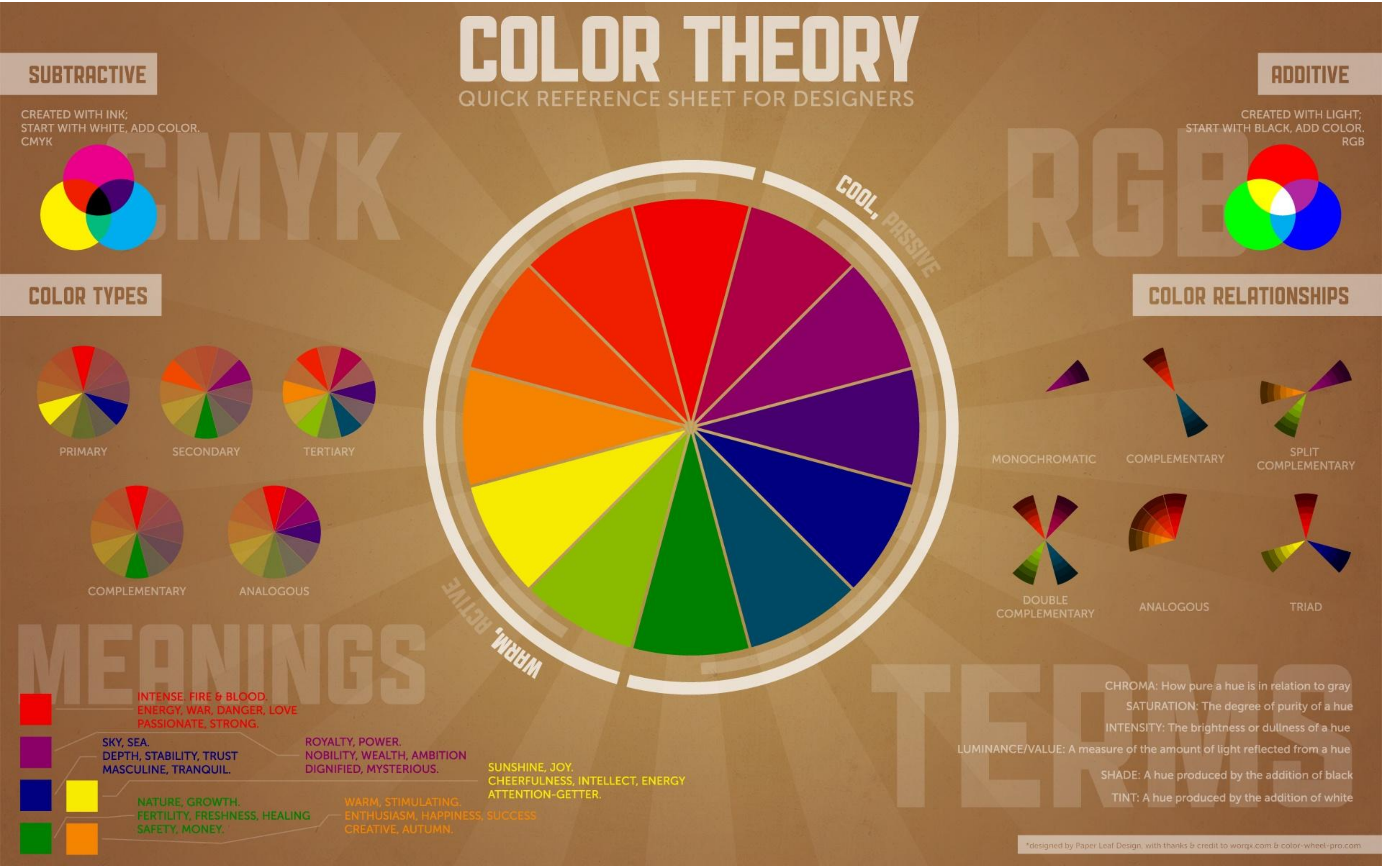
SPLIT COMPLEMENT

A COLOR AND THE TWO COLORS NEXT TO ITS COMPLEMENT ON THE COLOR WHEEL

YELLOW	BLUE VIOLET	RED VIOLET
YELLOW GREEN	VIOLET	RED
GREEN	RED VIOLET	RED ORANGE
BLUE GREEN	RED	ORANGE
BLUE	RED ORANGE	YELLOW ORANGE
BLUE VIOLET	ORANGE	YELLOW
VIOLET	YELLOW ORANGE	YELLOW GREEN
RED VIOLET	YELLOW	GREEN
RED	YELLOW GREEN	BLUE GREEN
RED ORANGE	GREEN	BLUE
ORANGE	BLUE GREEN	BLUE VIOLET
YELLOW ORANGE	BLUE	VIOLET

Color Wheel

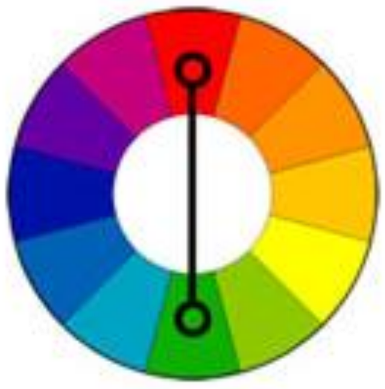
Source: http://graf1x.com/color-wheel-history-and-explanation/

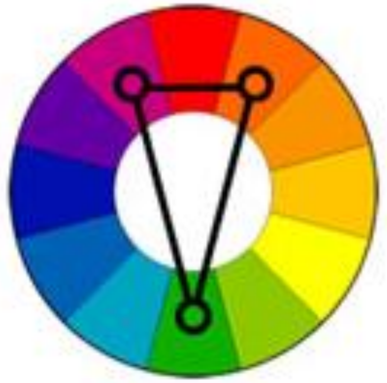

COLOR THEORY
QUICK REFERENCE SHEET FOR DESIGNERS
SUBTRACTIVE
CREATED WITH INK; START WITH WHITE, ADD COLOR. CMYK
CMYK
ADDITIVE
CREATED WITH LIGHT; START WITH BLACK, ADD COLOR. RGB
RGB
COOL, PASSIVE
WARM, ACTIVE
COLOR TYPES
PRIMARY
SECONDARY
TERTIARY
COMPLEMENTARY
ANALOGOUS
COLOR RELATIONSHIPS
MONOCHROMATIC
COMPLEMENTARY
SPLIT COMPLEMENTARY
DOUBLE COMPLEMENTARY
ANALOGOUS
TRIAD
MEANINGS
INTENSE. FIRE & BLOOD. ENERGY, WAR, DANGER, LOVE PASSIONATE, STRONG.
SKY, SEA. DEPTH, STABILITY, TRUST MASCULINE, TRANQUIL.
ROYALTY, POWER. NOBILITY, WEALTH, AMBITION DIGNIFIED, MYSTERIOUS.
SUNSHINE, JOY. CHEERFULNESS, INTELLECT, ENERGY ATTENTION-GETTER.
NATURE, GROWTH. FERTILITY, FRESHNESS, HEALING SAFETY, MONEY.
WARM, STIMULATING. ENTHUSIASM, HAPPINESS, SUCCESS CREATIVE, AUTUMN.
TERMS
CHROMA: How pure a hue is in relation to gray
SATURATION: The degree of purity of a hue
INTENSITY: The brightness or dullness of a hue
LUMINANCE/VALUE: A measure of the amount of light reflected from a hue
SHADE: A hue produced by the addition of black
TINT: A hue produced by the addition of white
*designed by Paper Leaf Design, with thanks & credit to worqx.com & color-wheel-pro.com

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

Colour Harmonies

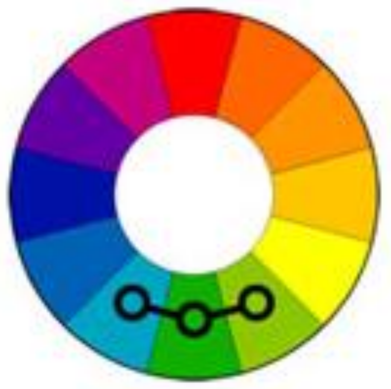
- **Complementary** – opposite colors on the color wheel; high contrast creates a vibrant look especially at full saturation
- **Split-complementary** – base color + two colors adjacent to its complement

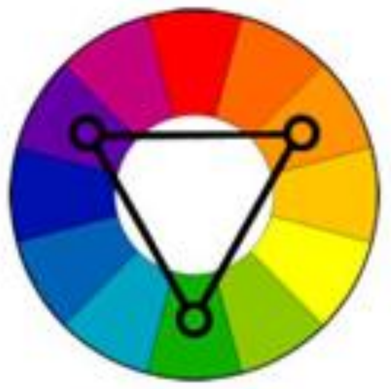

Complementary


Split-Complementary

Colour Harmonies

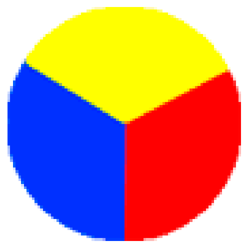
- **Analogous** – colors that are next to each other; pleasing, harmonious, create serene and comfortable design
- **Triadic** – colors evenly spaced around the color wheel; quite vibrant even if unsaturated

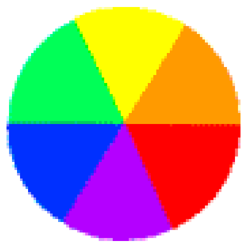

Analogous

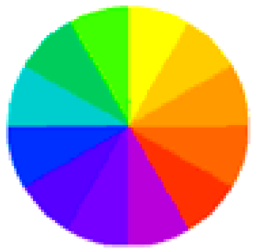

Triadic

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.

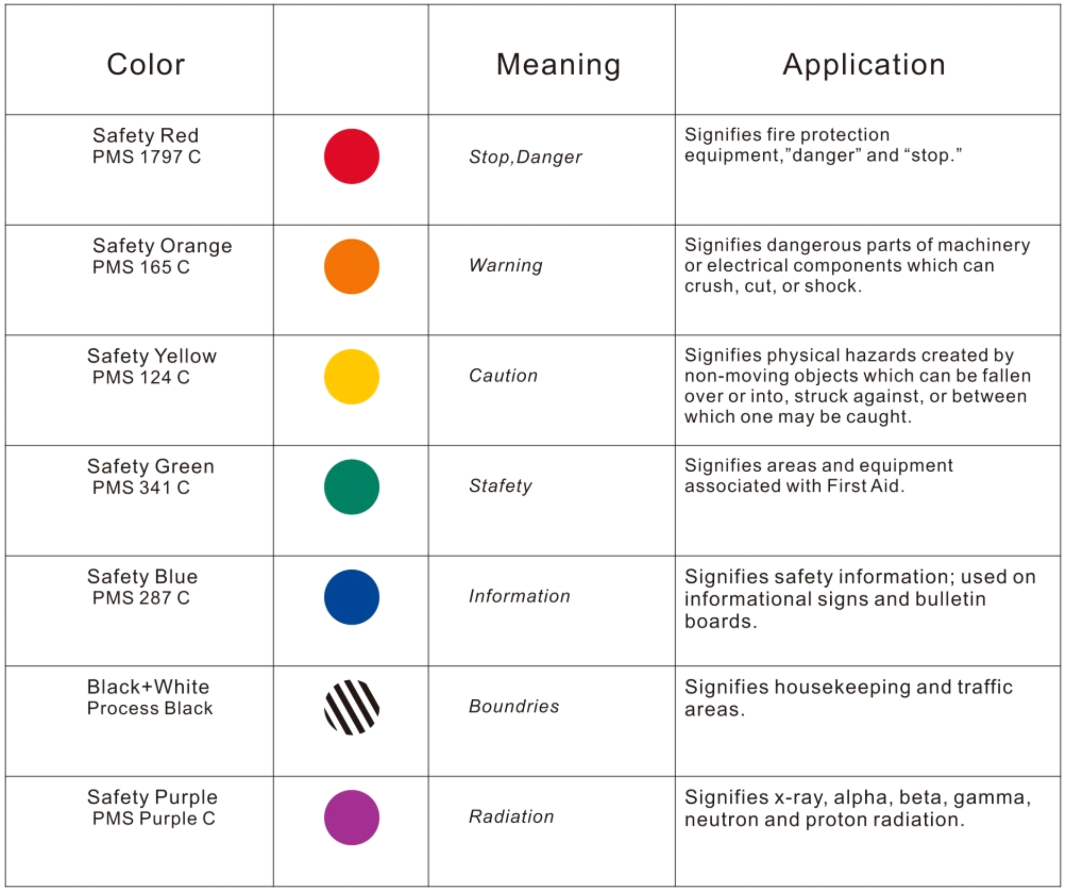
- **Hue** - the name of the color
- **Intensity** - the brightness or dullness of a color
- **Color value** - the darkness or lightness of a color (e.g. pink is a tint of red)
- **Tints** - are created by adding white to a color
- **Shades** - are created by adding black to a color
- **Optical color**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

Color Principles

- UI design should be recognizable and understable under various environments
- Perception of colors is highly subjective, but certain principles could be followed (see next slide)
- 70 % contrast between an object and its background makes the message most legible
- The typography in HMIs should ensure that the information is recognizable and understable even it becomes blurred or in dark environment
- Sans-serif styles work better than serif styles due to thin stroke

Color Principles

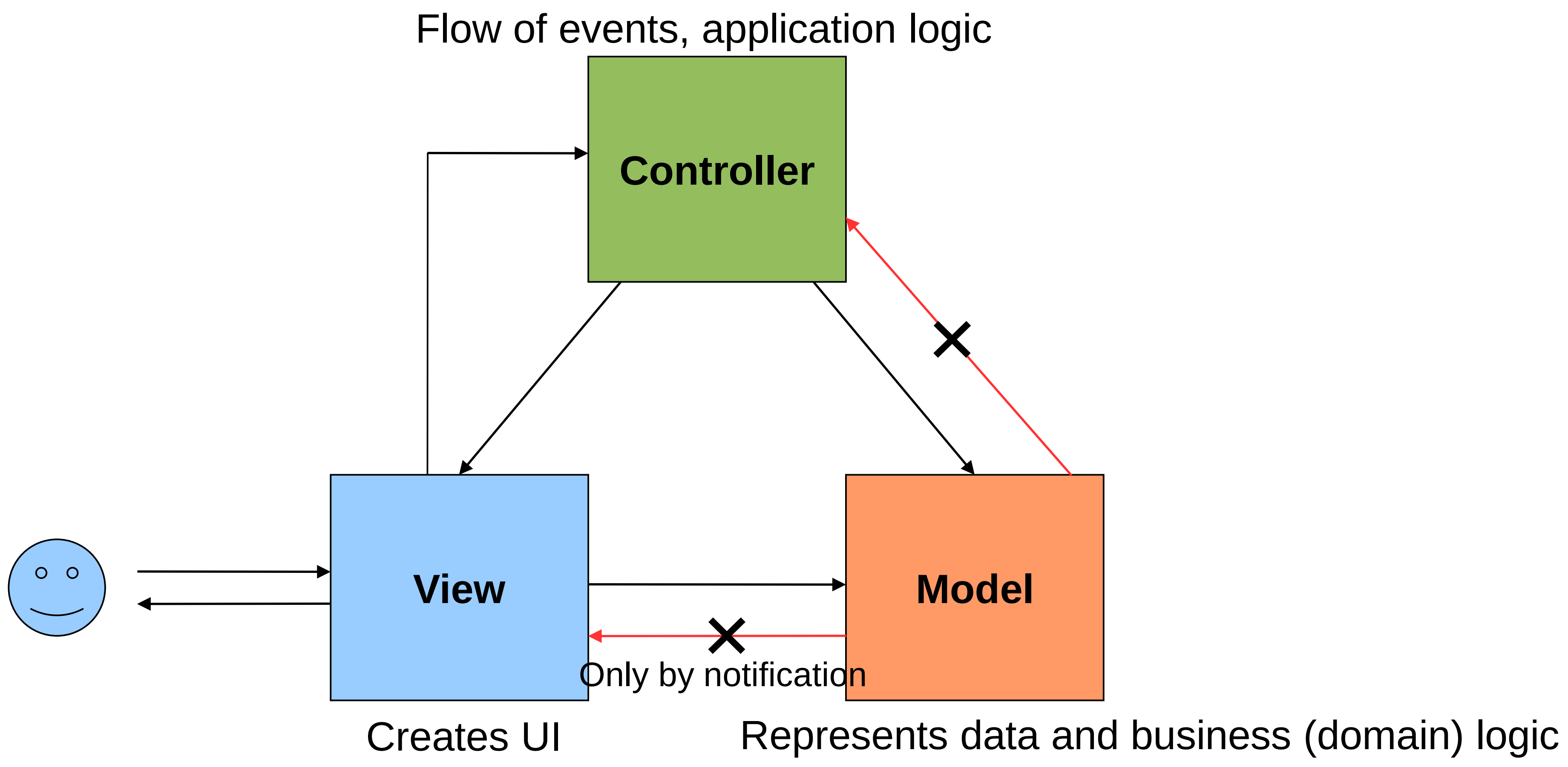
- UI design should be recognizable and understable under various environments
- Perception of colors is highly subjective, but certain principles could be followed



Color		Meaning	Application
Safety Red PMS 1797 C	🔴	*Stop,Danger*	Signifies fire protection equipment,"danger" and "stop."
Safety Orange PMS 165 C	🟠	*Warning*	Signifies dangerous parts of machinery or electrical components which can crush, cut, or shock.
Safety Yellow PMS 124 C	🟡	*Caution*	Signifies physical hazards created by non-moving objects which can be fallen over or into, struck against, or between which one may be caught.
Safety Green PMS 341 C	🟢	*Stafety*	Signifies areas and equipment associated with First Aid.
Safety Blue PMS 287 C	🔵	*Information*	Signifies safety information; used on informational signs and bulletin boards.
Black+White Process Black	▨	*Boundries*	Signifies housekeeping and traffic areas.
Safety Purple PMS Purple C	🟣	*Radiation*	Signifies x-ray, alpha, beta, gamma, neutron and proton radiation.

Model-View-Controller Architecture

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


Flow of events, application logic
Controller
View
Model
Only by notification
Creates UI
Represents data and business (domain) logic

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

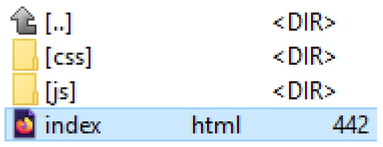
  https://getbootstrap.com/docs/5.3/getting-started/introduction

  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



```
[..]                <DIR>
[css]               <DIR>
[js]                <DIR>
index      html       442
```

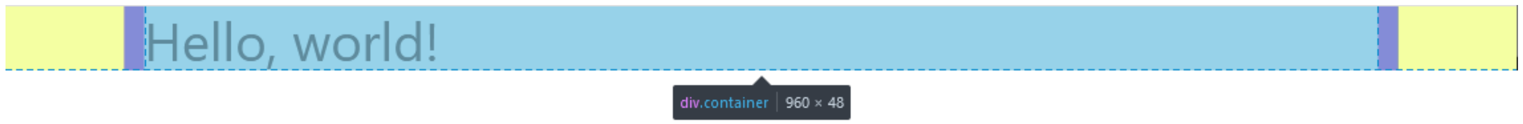
https://github.com/twbs/bootstrap/releases/download/v5.3.3/bootstrap-5.3.3-dist.zip

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)

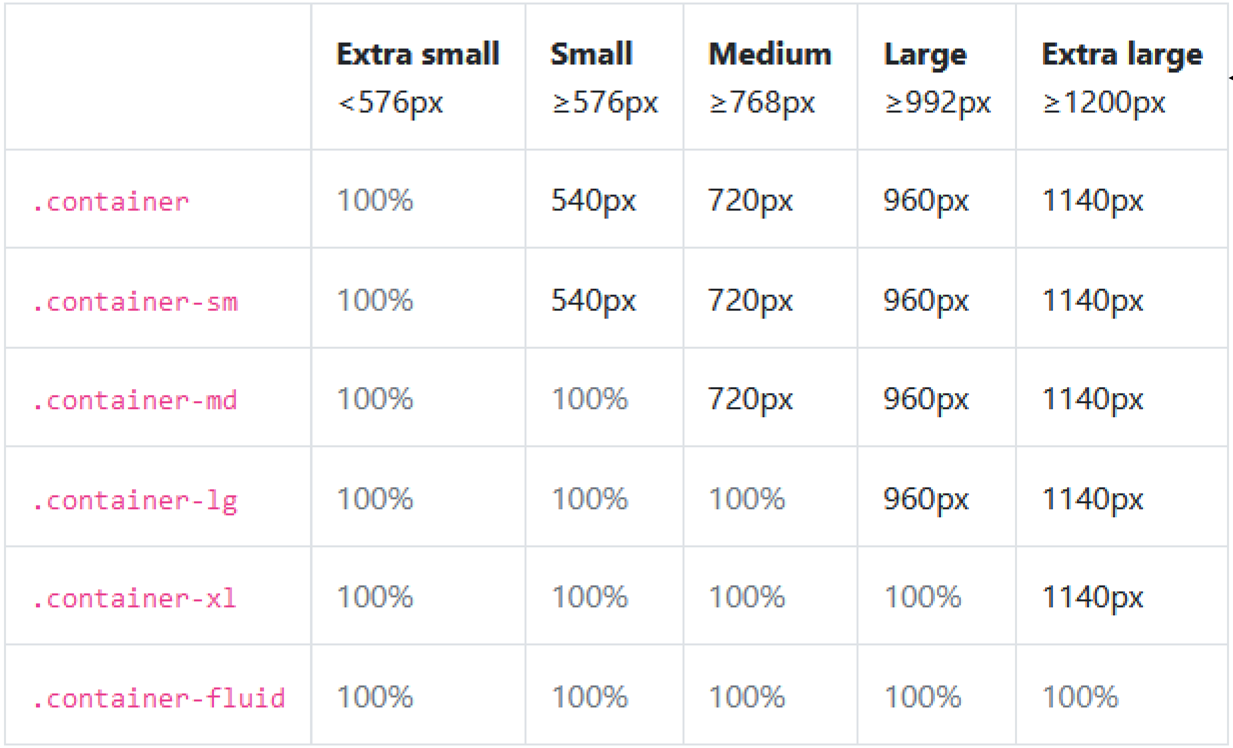
```
<div class="container">
  <h1>Hello, world!</h1>
</div>
```

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



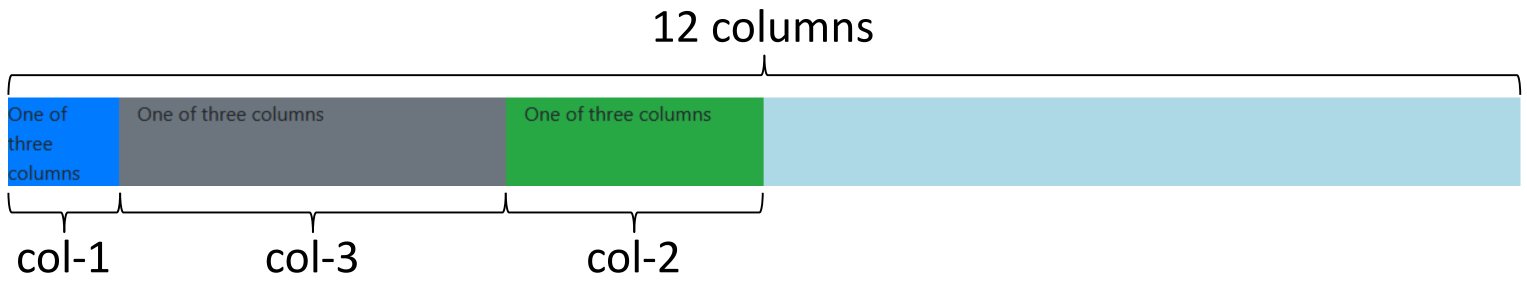
	Extra small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	Extra large ≥1200px
`.container`	100%	540px	720px	960px	1140px
`.container-sm`	100%	540px	720px	960px	1140px
`.container-md`	100%	100%	720px	960px	1140px
`.container-lg`	100%	100%	100%	960px	1140px
`.container-xl`	100%	100%	100%	100%	1140px
`.container-fluid`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

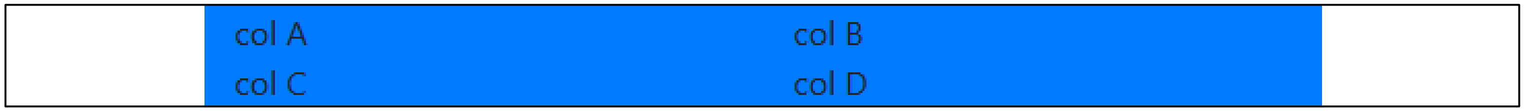
- Maximum number of columns in a single row is 12
- Column classes indicate the number of columns you'd like to use out of the possible 12 per row. If you want three equal-width columns across, you can use col-4 (i.e. number 4 represents the columnspan parameter)



```
<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

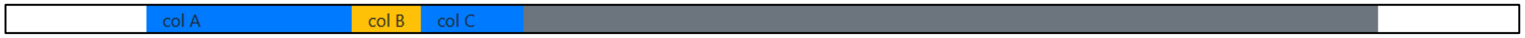
- Create equal-width columns that span multiple lines by inserting a w-100 where you want the columns to break to a new line



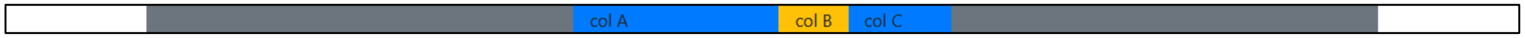
```
<div class="container bg-primary">
  <div class="row">
    <div class="col">col A</div>
    <div class="col">col B</div>
    **<div class="w-100"></div> <!-- break line -->**
    <div class="col">col C</div>
    <div class="col">col D</div>
  </div>
</div>
```

Bootstrap Grids

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



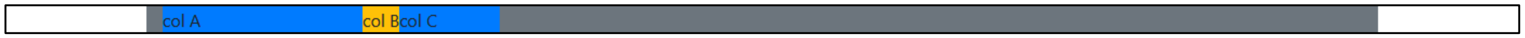
```
<div class="container bg-secondary">
  <div class="row justify-content-lg-start ">
    <div class="col-lg-2 bg-primary">col A</div>
    <div class="col-md-auto bg-warning">col B</div>
    <div class="col-lg-1 bg-primary">col C</div>
  </div>
</div>
```



```
<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)

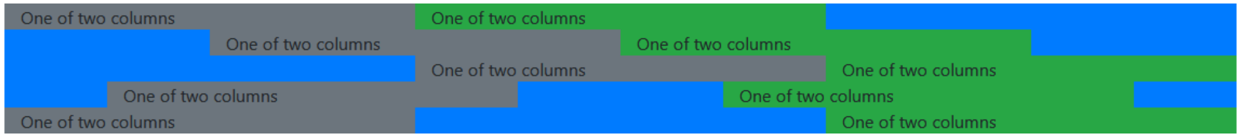


```
<div class="container bg-secondary">
  <div class="row justify-content-lg-start g-0">
    <div class="col-lg-2 bg-primary">col A</div>
    <div class="col-md-auto bg-warning">col B</div>
    <div class="col-lg-1 bg-primary">col C</div>
  </div>
</div>
```

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

Bootstrap Grids

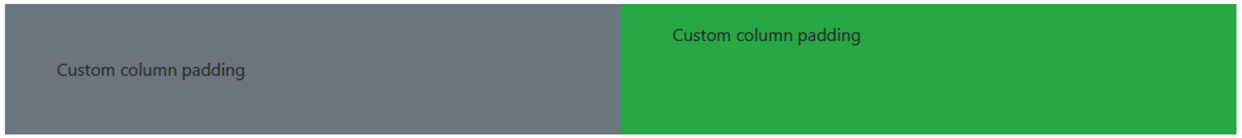
- Horizontal alignment



```
<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

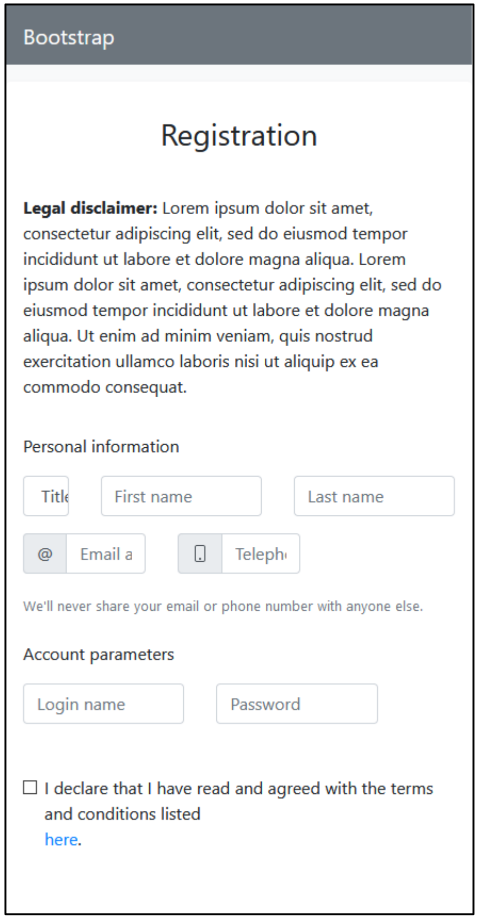
- Horizontal and vertical padding



```
<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



Bootstrap

Registration

Legal disclaimer: Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Personal information

Titl | First name | Last name

@ Email a | Teleph

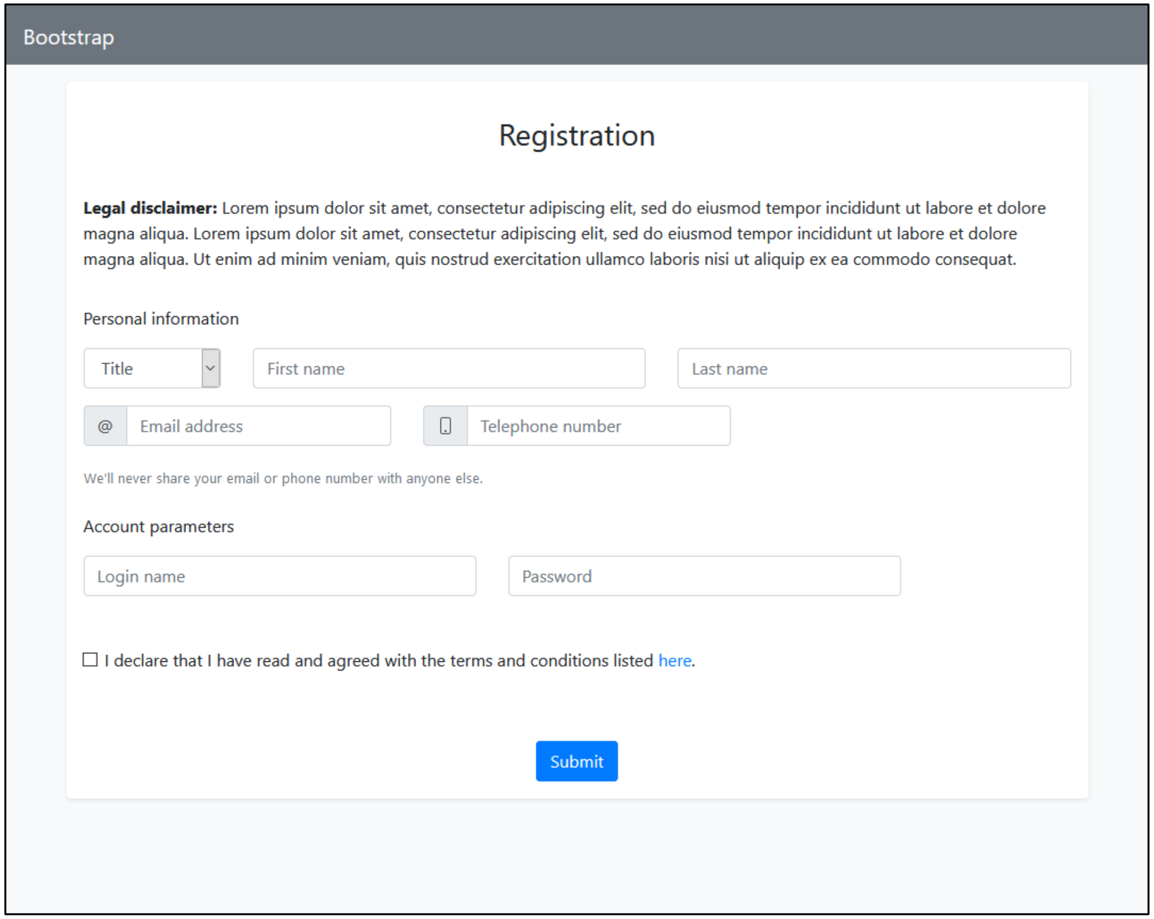
We'll never share your email or phone number with anyone else.

Account parameters

Login name | Password

☐ I declare that I have read and agreed with the terms and conditions listed here.

Small display



Bootstrap

Registration

Legal disclaimer: Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Personal information

Title | First name | Last name

@ Email address | Telephone number

We'll never share your email or phone number with anyone else.

Account parameters

Login name | Password

☐ I declare that I have read and agreed with the terms and conditions listed here.

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/

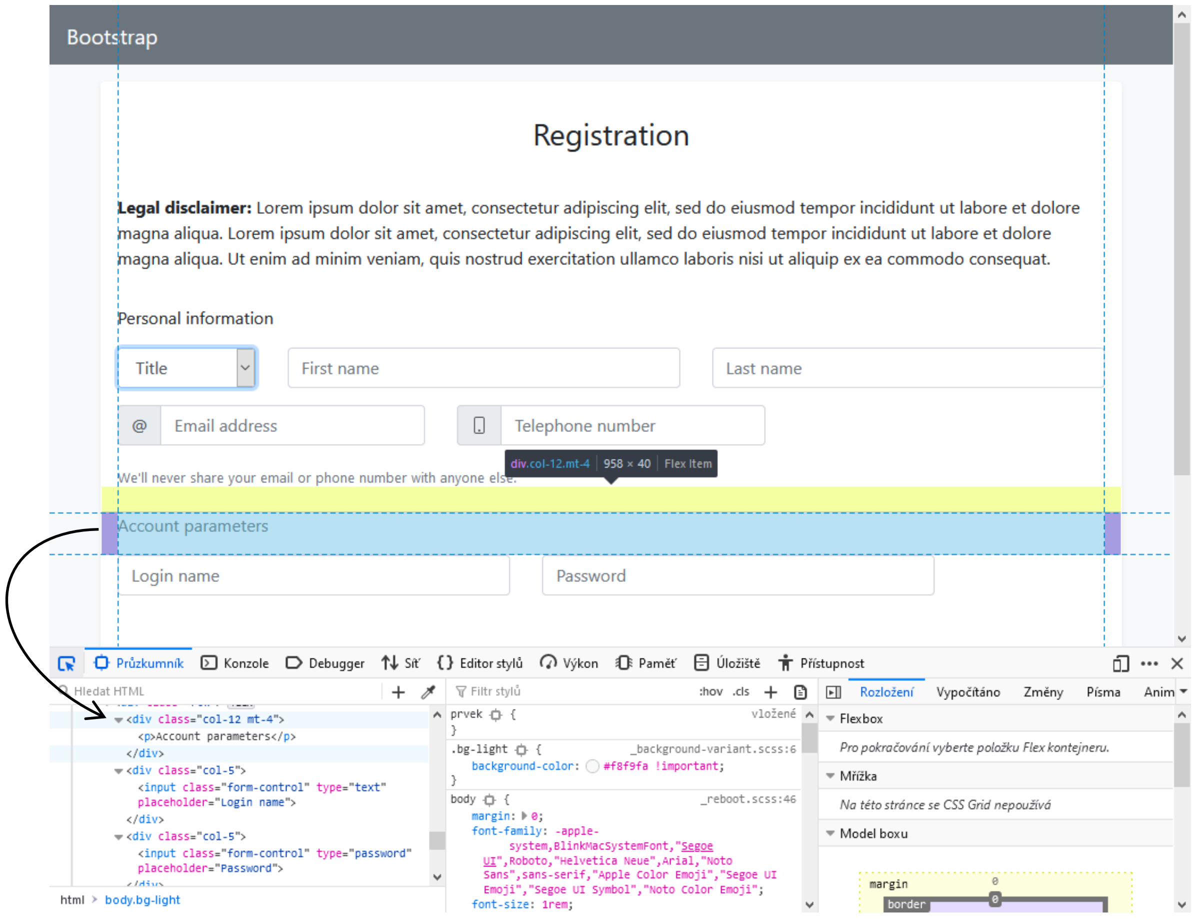
- Also see the example below for a better idea of how it all works

  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)



Registration

Legal disclaimer: Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Personal information

Title | First name | Last name

@ Email address | Telephone number

We'll never share your email or phone number with anyone else.

Account parameters

Login name | Password

```
<div class="col-12 mt-4">
  <p>Account parameters</p>
</div>
<div class="col-5">
  <input class="form-control" type="text"
  placeholder="Login name">
</div>
<div class="col-5">
  <input class="form-control" type="password"
  placeholder="Password">
```

```
prvek {                                   vložené
}
.bg-light {          _background-variant.scss:6
  background-color: #f8f9fa !important;
}
body {                          _reboot.scss:46
  margin: 0;
  font-family: -apple-
      system,BlinkMacSystemFont,"Segoe
   UI",Roboto,"Helvetica Neue",Arial,"Noto
   Sans",sans-serif,"Apple Color Emoji","Segoe UI
   Emoji","Segoe UI Symbol","Noto Color Emoji";
  font-size: 1rem;
```

Flexbox

Pro pokračování vyberte položku Flex kontejneru.

Mřížka

Na této stránce se CSS Grid nepoužívá

Model boxu

Dashboard Collection

Aston Martin DB9

BMW i8

BMW 7 Comfort Mode

VW Cross Blue Coupe Concept

KIA K900

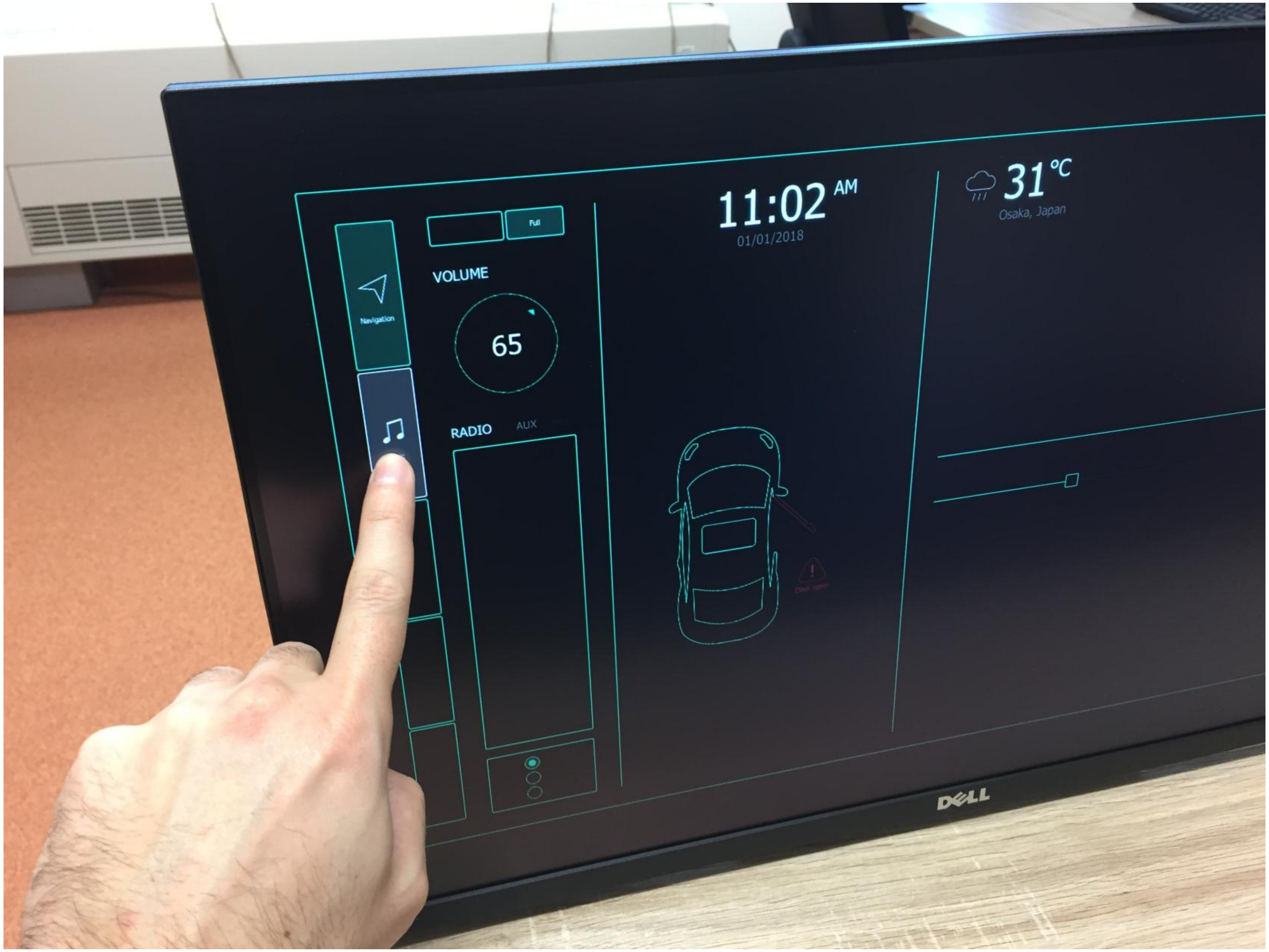
Source: https://medium.com/@dnevozhai/car-dashboard-ui-collection-123ce3ab5303

Dashboard Collection

Ford Mustang

Dashboard in Qt 3D

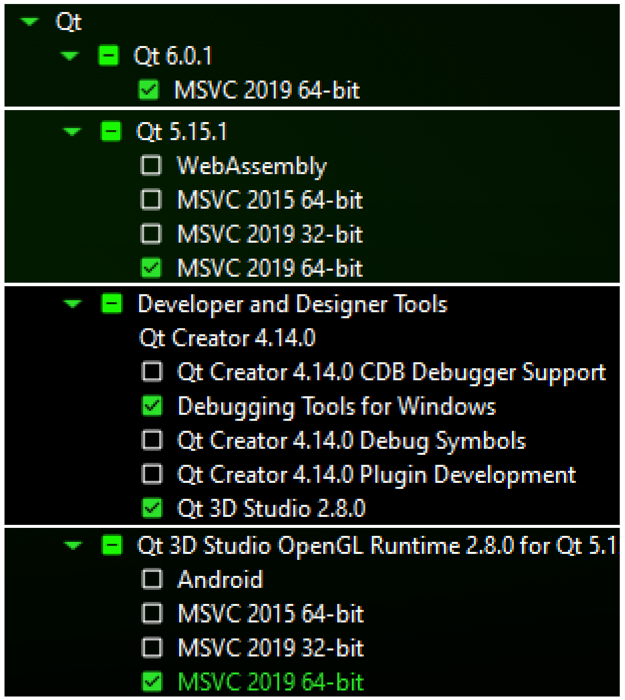
Dashboard in Qt 3D



Qt 5 (and 6) Installation

Visual Studio 2022 (Cummunity Edition) with package "Developing desktop applications using C++" is expected to be already installed

- Download qt-unified-windows-x86-4.7.0-online.exe
- Run it and log in to your Qt account or create a new one
- Check the obligations and check the individual development
- Disable sending pseudonymous usage stats
- Select custom installation as follows
  - Qt **5.15.1** for **MSVC 2019 64-bit**
  - Qt 6.0.2 for MSVC 2019 64-bit (optional)
  - Qt 3D Studio **2.8.0**
  - Qt 3D Studio OpenGL Runtime **2.8.0** for **Qt 5.15.1** and **MSVC 2019 64-bit**
- API documentation http://doc.qt.io/



```
Qt
  Qt 6.0.1
    [✓] MSVC 2019 64-bit
  Qt 5.15.1
    [ ] WebAssembly
    [ ] MSVC 2015 64-bit
    [ ] MSVC 2019 32-bit
    [✓] MSVC 2019 64-bit
  Developer and Designer Tools
    Qt Creator 4.14.0
    [ ] Qt Creator 4.14.0 CDB Debugger Support
    [✓] Debugging Tools for Windows
    [ ] Qt Creator 4.14.0 Debug Symbols
    [ ] Qt Creator 4.14.0 Plugin Development
    [✓] Qt 3D Studio 2.8.0
  Qt 3D Studio OpenGL Runtime 2.8.0 for Qt 5.1
    [ ] Android
    [ ] MSVC 2015 64-bit
    [ ] MSVC 2019 32-bit
    [✓] MSVC 2019 64-bit
```

Qt 5 Installation

1. Qt Installer (Maintenance Tool 4.8.1)

   https://download.qt.io/official_releases/online_installers/qt-unified-windows-x64-online.exe

2. Qt 5.15.1 MSVC 2019 64bit (refer to the next slide)

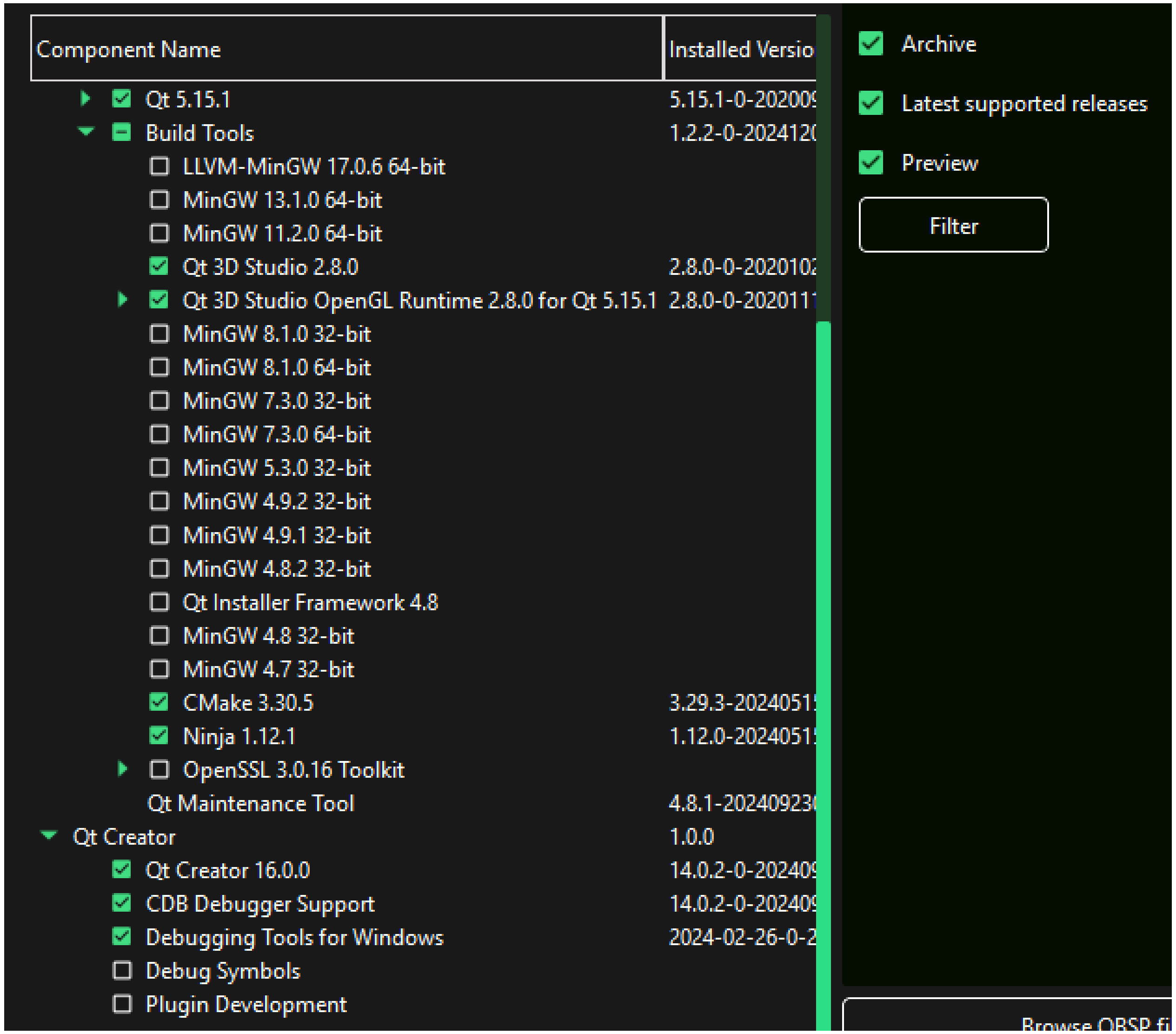
3. Qt VS Tools Extension

4a. Qt 3D Studio 2.8 for Qt 5.15.1 MSVC 2019 (only if not selected during the Qt installation)

   https://download.qt.io/official_releases/qt3dstudio/2.8/qt-3dstudio-opensource-windows-x86-2.8.0.exe

4b. Copy *.* from c:\Qt\Qt3DStudio-2.8.0\Tools\Qt3DStudio\ to c:\Qt\5.15.1\msvc2019_64\ and keep the existing files

Qt 5 Installation



Component Name	Installed Versio
▸ ✓ Qt 5.15.1	5.15.1-0-20200
▾ ▬ Build Tools	1.2.2-0-202412
☐ LLVM-MinGW 17.0.6 64-bit	
☐ MinGW 13.1.0 64-bit	
☐ MinGW 11.2.0 64-bit	
✓ Qt 3D Studio 2.8.0	2.8.0-0-202010
▸ ✓ Qt 3D Studio OpenGL Runtime 2.8.0 for Qt 5.15.1	2.8.0-0-202011
☐ MinGW 8.1.0 32-bit	
☐ MinGW 8.1.0 64-bit	
☐ MinGW 7.3.0 32-bit	
☐ MinGW 7.3.0 64-bit	
☐ MinGW 5.3.0 32-bit	
☐ MinGW 4.9.2 32-bit	
☐ MinGW 4.9.1 32-bit	
☐ MinGW 4.8.2 32-bit	
☐ Qt Installer Framework 4.8	
☐ MinGW 4.8 32-bit	
☐ MinGW 4.7 32-bit	
✓ CMake 3.30.5	3.29.3-2024051
✓ Ninja 1.12.1	1.12.0-2024051
▸ ☐ OpenSSL 3.0.16 Toolkit	
Qt Maintenance Tool	4.8.1-2024092
▾ Qt Creator	1.0.0
✓ Qt Creator 16.0.0	14.0.2-0-20240
✓ CDB Debugger Support	14.0.2-0-20240
✓ Debugging Tools for Windows	2024-02-26-0-2
☐ Debug Symbols	
☐ Plugin Development	

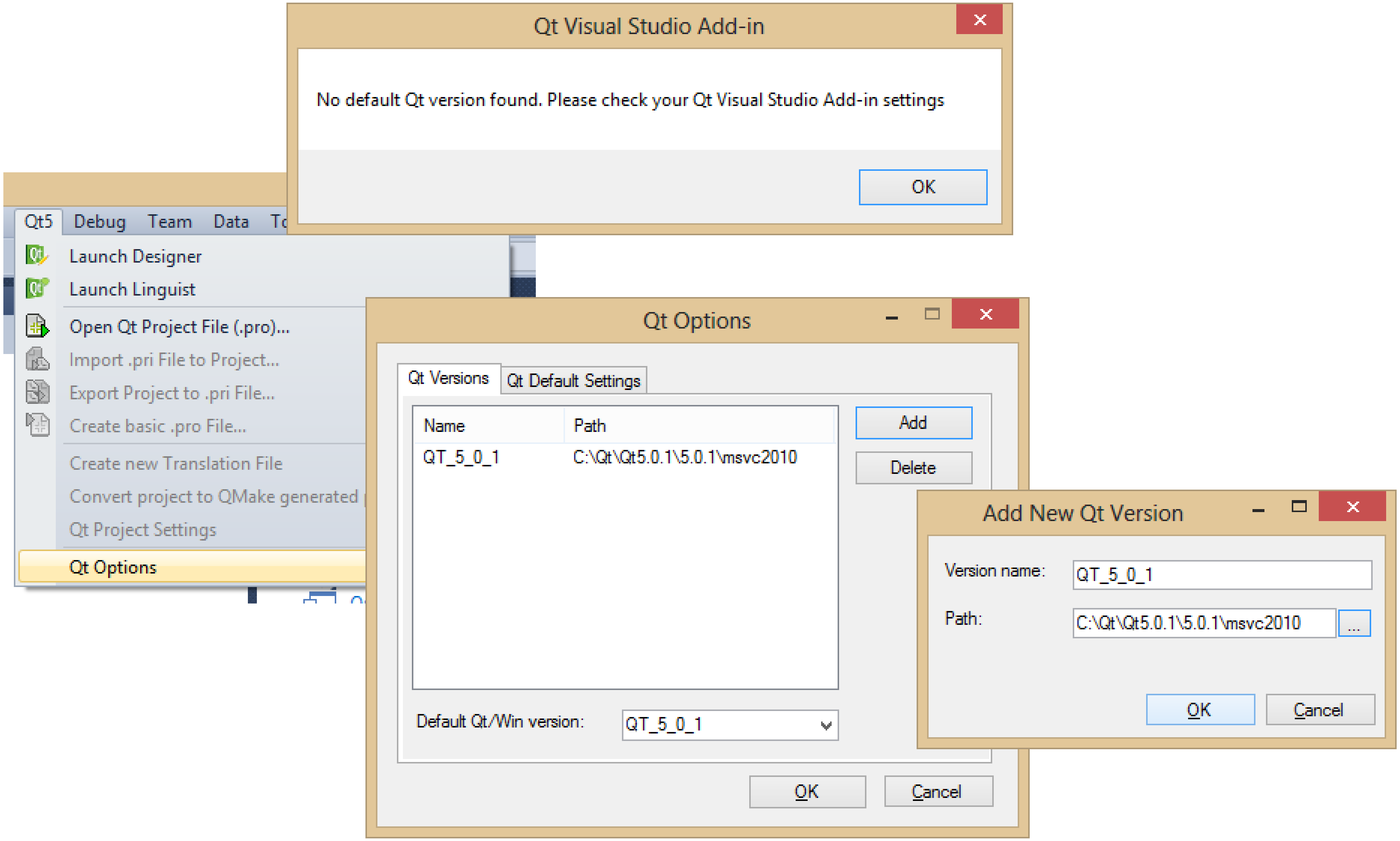
✓ Archive

✓ Latest supported releases

✓ Preview

Qt Version Setup

Firstly, setup the Qt environment in VS (note that versions and paths may vary).



Qt Visual Studio Add-in

No default Qt version found. Please check your Qt Visual Studio Add-in settings

Qt Options

Name	Path
QT_5_0_1	C:\Qt\Qt5.0.1\5.0.1\msvc2010

Default Qt/Win version: QT_5_0_1

Add New Qt Version

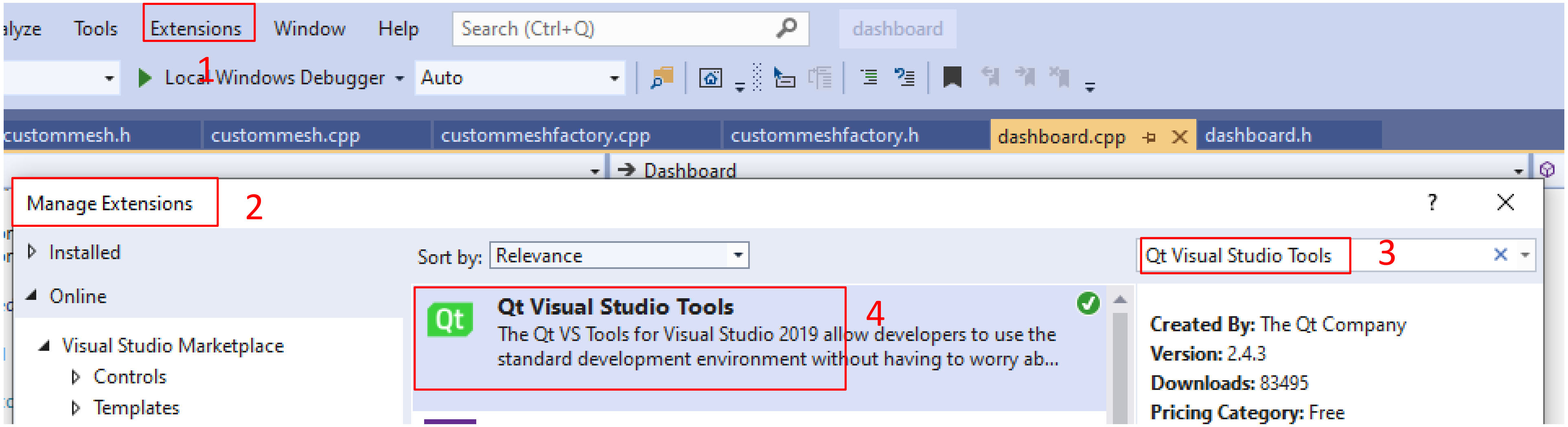
Version name: QT_5_0_1

Path: C:\Qt\Qt5.0.1\5.0.1\msvc2010

OK
Qt5
Debug
Team
Data
Launch Designer
Launch Linguist
Open Qt Project File (.pro)...
Import .pri File to Project...
Export Project to .pri File...
Create basic .pro File...
Create new Translation File
Convert project to QMake generated
Qt Project Settings
Qt Options
Qt Versions
Qt Default Settings
Add
Delete
OK
Cancel
...
OK
Cancel

Installation of Qt VS Tools Extension

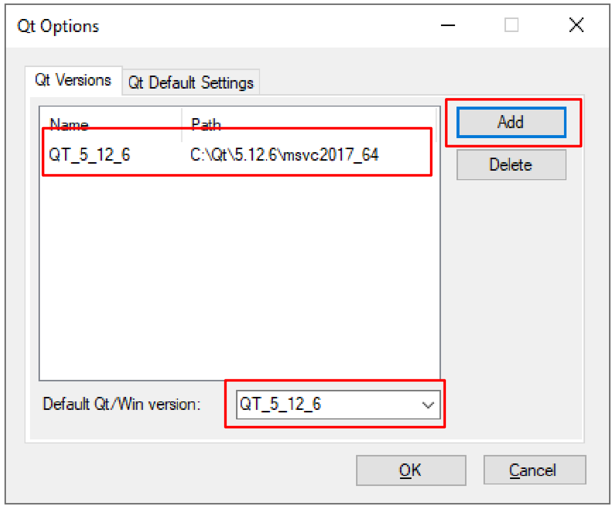
- We also need to install Qt Visual Studio Tools extension



Qt Visual Studio Tools
The Qt VS Tools for Visual Studio 2019 allow developers to use the standard development environment without having to worry ab...

Created By: The Qt Company
Version: 2.4.3
Downloads: 83495
Pricing Category: Free

- And set the correct path where the Qt library is installed



Name	Path
QT_5_12_6	C:\Qt\5.12.6\msvc2017_64

Default Qt/Win version: QT_5_12_6

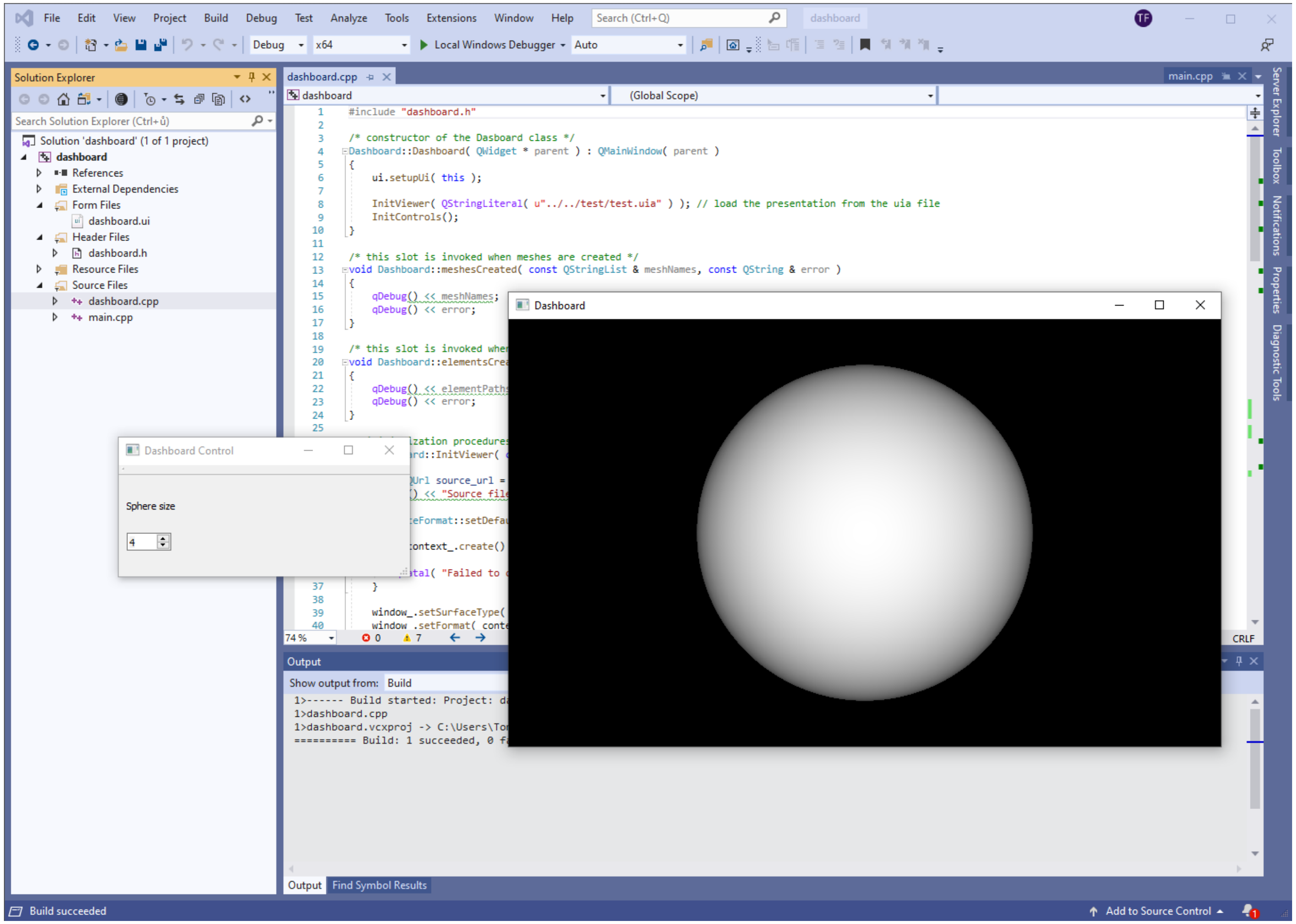
Towards the Dashboard in Qt

- Our dashboard will consist of two components
  - the presentation (created in Qt 3D Studio and stored in the .uip file)
  - the code in C++ (this will be done in VS C++ wit Qt library)
- In principle, the presentation will contain all items which can be done easily in Qt Studio (text labels, indicators, materials, complex geometry stored in fbx files, lighting etc.)
- In the code, we will create all stuff which is (or should be)
  - parametrical (e.g. geometry of speedometer ticks)
  - variable over time (e.g. position of speedometer needles, state of indicators) and controlled by signals from sensors all over the car

Before jumping right into the implementation of a dashboard, the reader is expected to become familiar with the Qt 3D Studio OpenGL Runtime 2.8.0 Manual
https://doc.qt.io/qt3dstudio/openglruntime/qt3dstudio-opengl-runtime-index.html

Dashboard Template

- The template on my web site should make it easier to start…

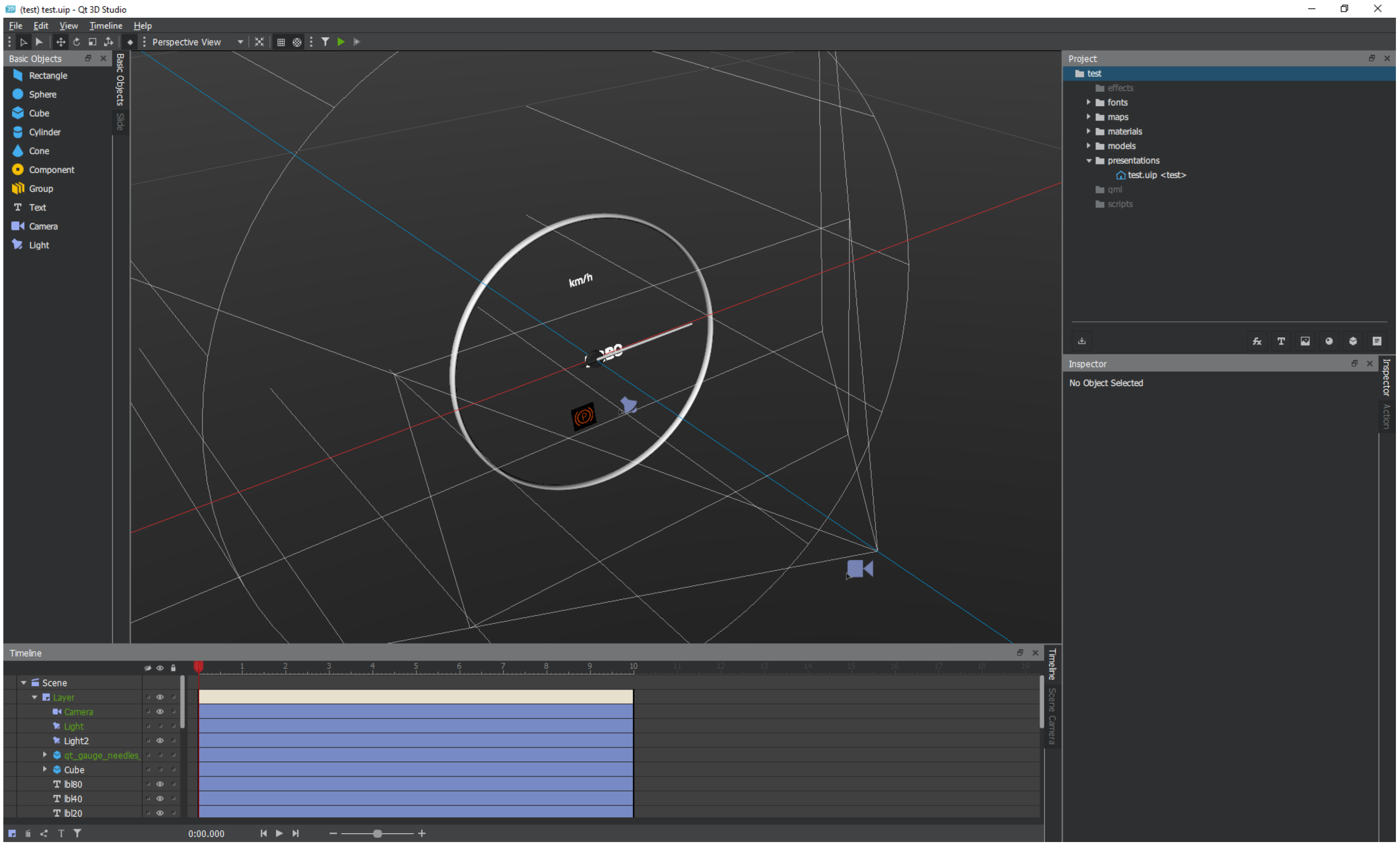


```
#include "dashboard.h"

/* constructor of the Dasboard class */
Dashboard::Dashboard( QWidget * parent ) : QMainWindow( parent )
{
    ui.setupUi( this );

    InitViewer( QStringLiteral( u"../../test/test.uia" ) ); // load the presentation from the uia file
    InitControls();
}

/* this slot is invoked when meshes are created */
void Dashboard::meshesCreated( const QStringList & meshNames, const QString & error )
{
    qDebug() << meshNames;
    qDebug() << error;
}

/* this slot is invoked whe
void Dashboard::elementsCrea
{
    qDebug() << elementPaths
    qDebug() << error;
}

         ization procedures
         rd::InitViewer(

         QUrl source_url =
         ) << "Source file

         eFormat::setDefau

         ontext_.create()

         tal( "Failed to
    }

    window_.setSurfaceType(
    window_.setFormat( conte
```

Show output from: Build

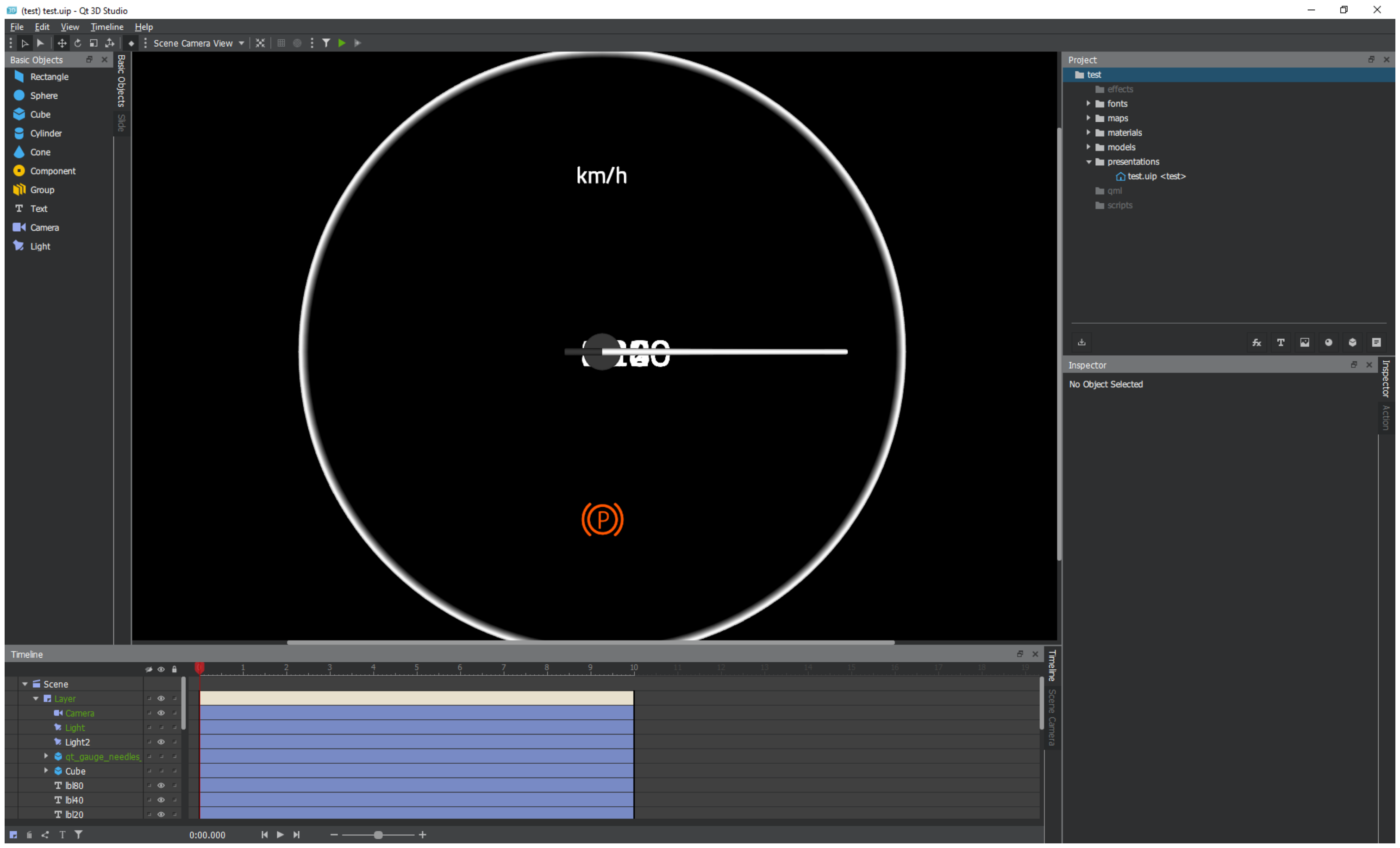
```
1>------ Build started: Project: da
1>dashboard.cpp
1>dashboard.vcxproj -> C:\Users\To
========== Build: 1 succeeded, 0 fa
```

Setting-up of Presentation



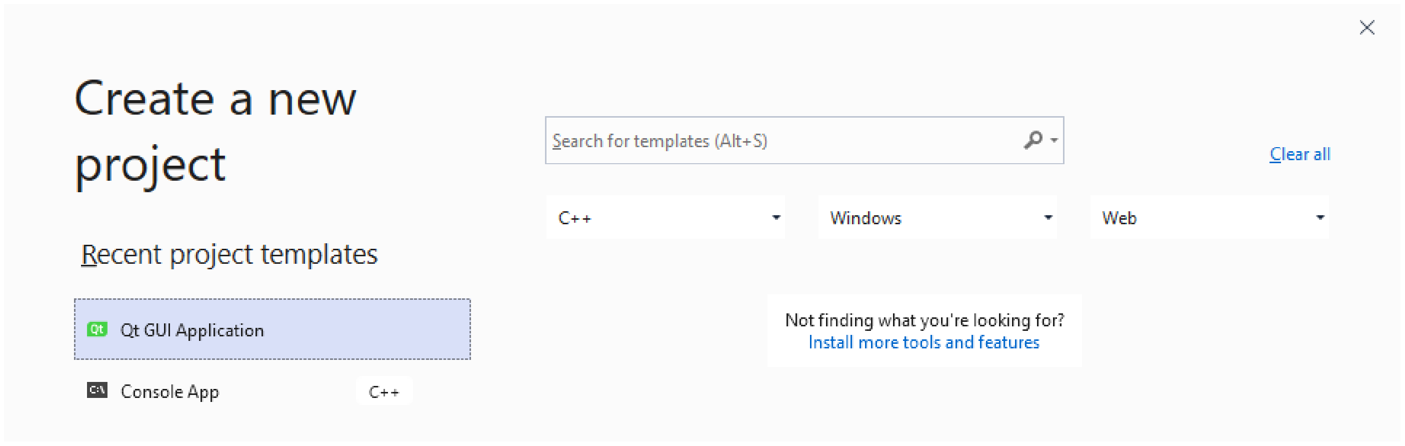
Setting-up of Presentation

- We need to layout all mentioned stuff in the field of view of our camera to see everything correctly in the preview window of the scene camera view



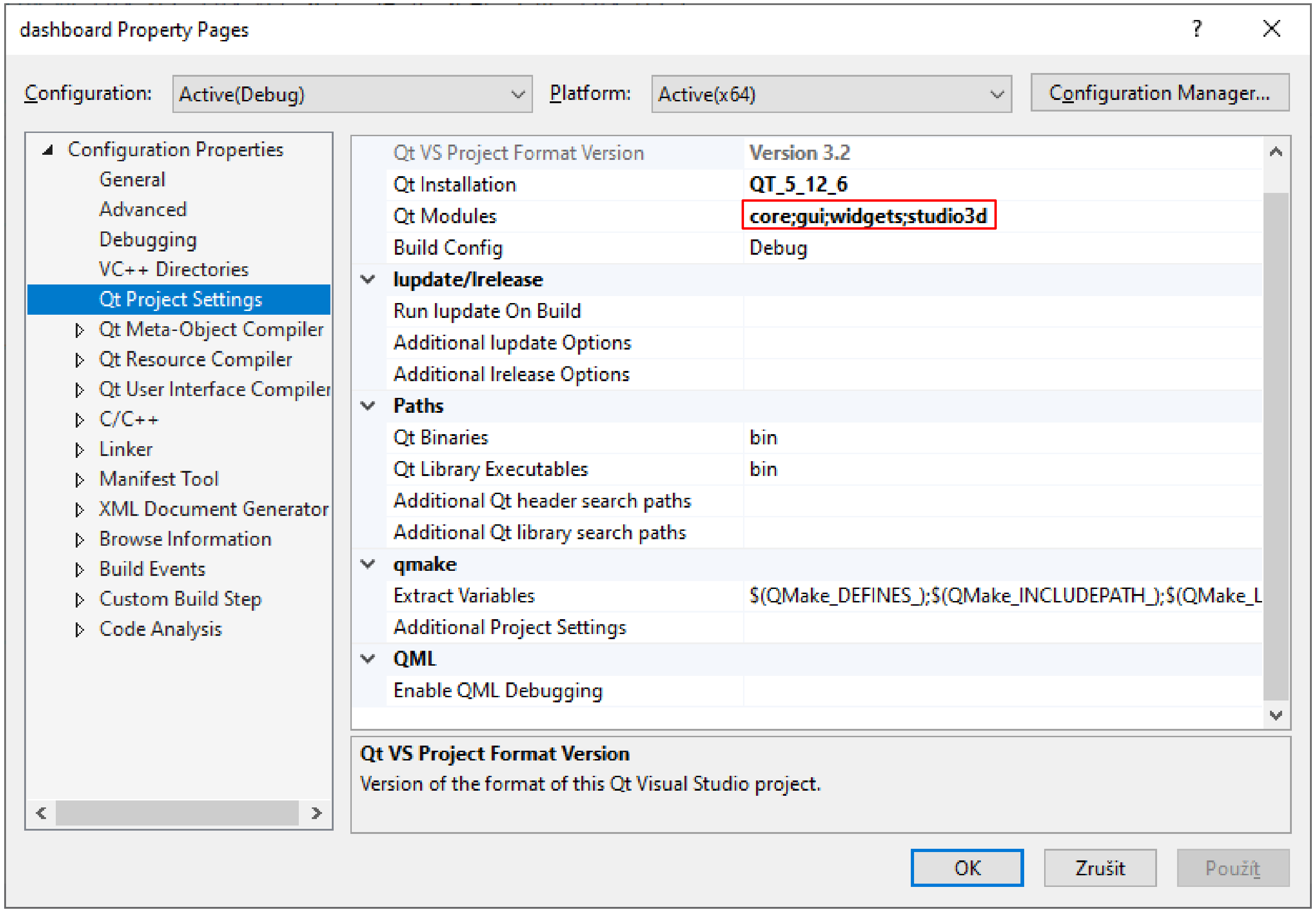
Setting-up of Code

- Now we need to create a new project in the Visual Studio (correctly installed Qt library and Qt plugin is assumed)
- We can start with Qt GUI Application template



Setting-up of Code

- Add **studio3d** module into the list of Qt Modules (dont forget to do it for both Release and Debug profiles)



dashboard Property Pages

Configuration: Active(Debug) Platform: Active(x64) Configuration Manager...

Qt VS Project Format Version	Version 3.2
Qt Installation	QT_5_12_6
Qt Modules	core;gui;widgets;studio3d
Build Config	Debug
lupdate/lrelease	
Run lupdate On Build	
Additional lupdate Options	
Additional lrelease Options	
Paths	
Qt Binaries	bin
Qt Library Executables	bin
Additional Qt header search paths	
Additional Qt library search paths	
qmake	
Extract Variables	$(QMake_DEFINES_);$(QMake_INCLUDEPATH_);$(QMake_L
Additional Project Settings	
QML	
Enable QML Debugging	

Qt VS Project Format Version
Version of the format of this Qt Visual Studio project.

See https://doc.qt.io/qt3dstudio/openglruntime/openglruntime-module.html

Control App Entry Point

- The main function looks very similar for all Qt applications. We can start with something like this in the main.cpp file

```cpp
#include "dashboard.h"
#include <QtWidgets/QApplication>

int main( int argc, char * argv[] )
{
  QCoreApplication::setAttribute( Qt::AA_EnableHighDpiScaling );

  QApplication application( argc, argv );

  Dashboard main_window; // our main window will contain everything
  main_window.setWindowTitle( "Dashboard Control" );
  main_window.show();

  return application.exec();
}
```

Dashboard Class

- Control panel for the dashboard will be realized in Dashboard class

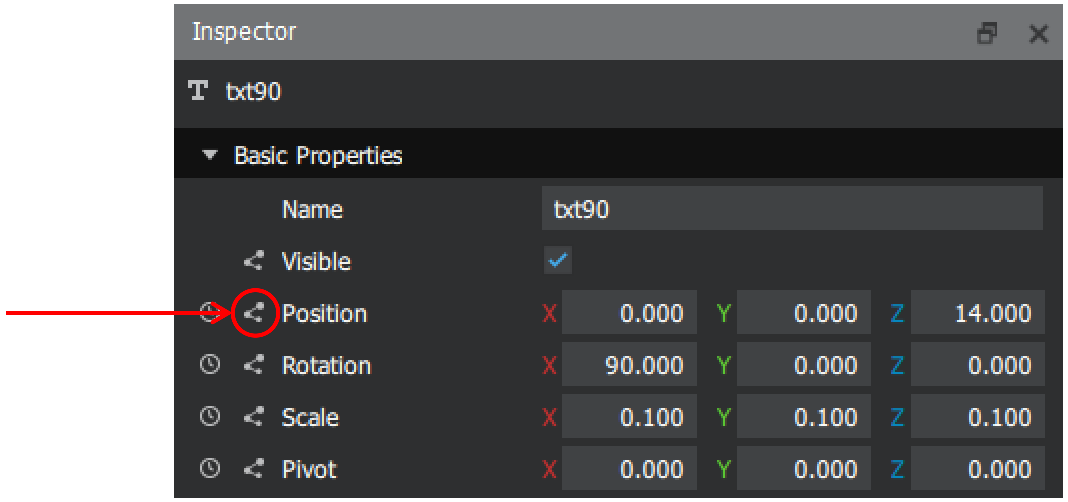
```
#pragma once

#include <QtWidgets/QMainWindow>
#include <QWindow>
#include <QtStudio3D>
#include "ui_dashboard.h"

class Dashboard : public QMainWindow
{
  Q_OBJECT
public:
  Dashboard( QWidget * parent = Q_NULLPTR );
  int InitViewer( const QString & source_file_name );
  int InitControls();
public slots:
  void meshesCreated( const QStringList & meshNames, const QString & error );
  void elementsCreated( const QStringList & elementPaths, const QString & error );
private:
  Ui::DashboardClass ui;
  QOpenGLContext context_;
  QWindow window_;
  Q3DSSurfaceViewer viewer_;
};
```

Parameters Control From C++

- Now we need to return briefly into the Qt 3D Studio to add some data inputs (on the following three slides) that allow us to control selected properties of Qt 3D Studio objects from the C++ code
- At least, we need to be able to
  - rotate the speedometer needles
  - switch the indicators on and off

Parameters Control From C++

- We can use DataInput to control selected attributes (e.g. position and rotation) of elements in the presentation
- You might by tempted to use 3DSPresentation::setAttribute method instead, but this surprisingly doesn't work
- To do so, first click on the „Set Data Input controller“ icon next to the property of the object you want to control from the code


Inspector

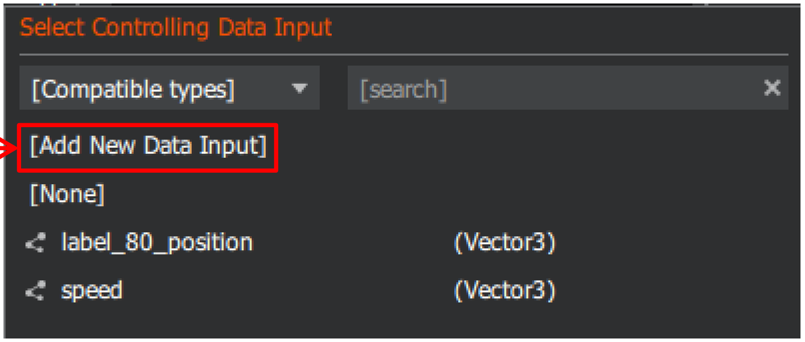
txt90

Basic Properties

Name	txt90					
Visible	✓					
Position	X	0.000	Y	0.000	Z	14.000
Rotation	X	90.000	Y	0.000	Z	0.000
Scale	X	0.100	Y	0.100	Z	0.100
Pivot	X	0.000	Y	0.000	Z	0.000

Parameters Control From C++

- A new window will popup, click on „Add New Data Input“ and



Select Controlling Data Input

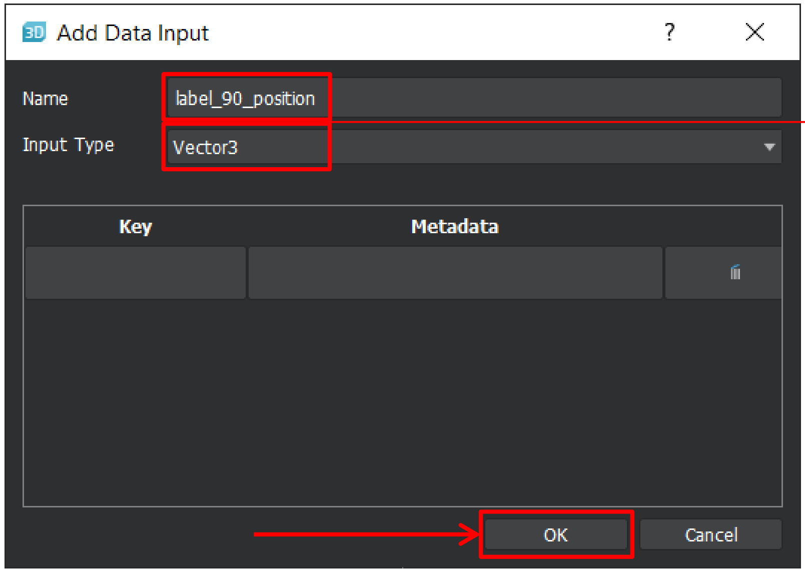
[Compatible types] [search]

[Add New Data Input]

[None]

label_80_position	(Vector3)
speed	(Vector3)

- fill the „Add Data Input“ form as needed



Add Data Input

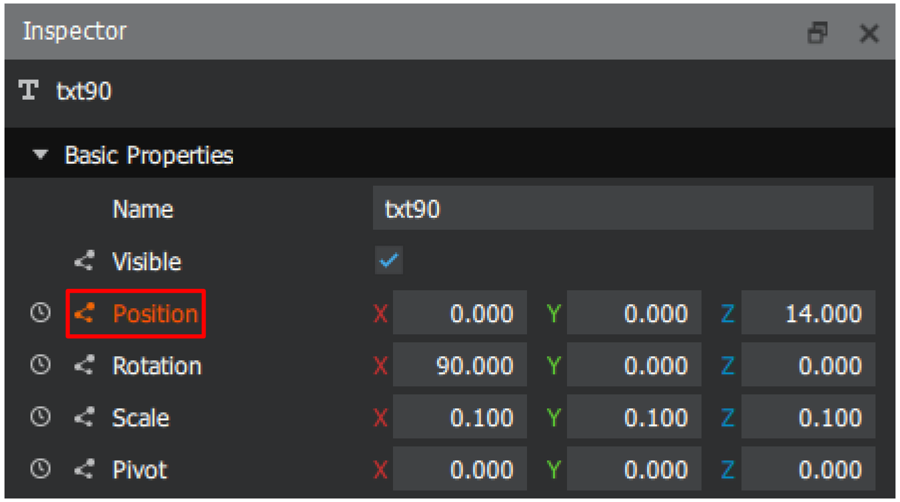
Name	label_90_position
Input Type	Vector3

Key	Metadata

The name and the datatype of DataInput we will be dealing with in the C++ code later. The name is arbitrary but the Type has to match the type of the property we are dealing with

Parameters Control From C++

- You should end up with something like this. Controlled property will be highlighted in orange



Inspector

txt90

Basic Properties

Name	txt90						
Visible	✓						
Position	X	0.000	Y	0.000	Z	14.000
Rotation	X	90.000	Y	0.000	Z	0.000
Scale	X	0.100	Y	0.100	Z	0.100
Pivot	X	0.000	Y	0.000	Z	0.000

- Save the project and move back to the code

Parameters Control From C++

- In the code, we need to get all the data inputs from the presentation

dashboard.cpp

```
// get the list of all data inputs created in the presentation
const auto dataInputs = viewer_.presentation()->dataInputs();

for ( const auto & it : dataInputs )
{
  Q3DSDataInput * dataInput = it;

  if ( dataInput->name() == QLatin1String( "label_90_position" ) &&
       dataInput->isValid() )
  {
    label90Position_ = dataInput;
  }
}
```

dashboard.h

```
private: Q3DSDataInput * label90Position_{ nullptr };
```

Parameters Control From C++

- Now, we can control various attributes of elements in the presentation directly from the code

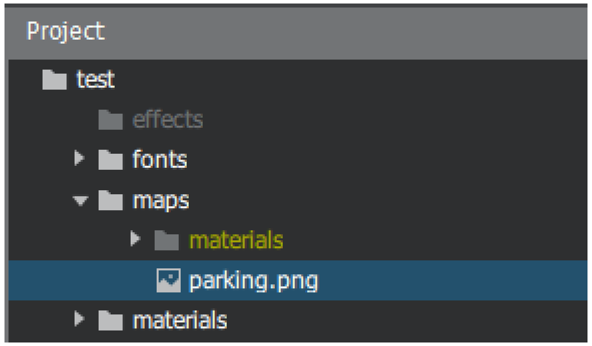
dashboard.cpp

```
// move the object to a new position
label90Position_->setValue( QVector3D( -3.0f, 0.0f, 13.0f ) );
```

How to Create Dashboard Icon

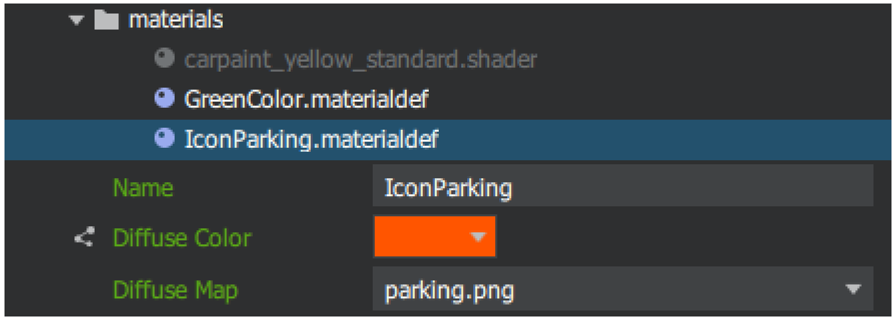
- Prepare icon image (~128×128 px, grayscale, png)

- Move the icon file into the maps folder



```
Project
test
  effects
  fonts
  maps
    materials
    parking.png
  materials
```

- Create a new (basic) material



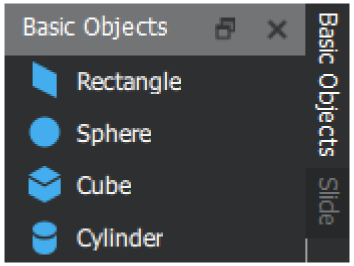
```
materials
  carpaint_yellow_standard.shader
  GreenColor.materialdef
  IconParking.materialdef
```

Name	IconParking
Diffuse Color	🟧
Diffuse Map	parking.png

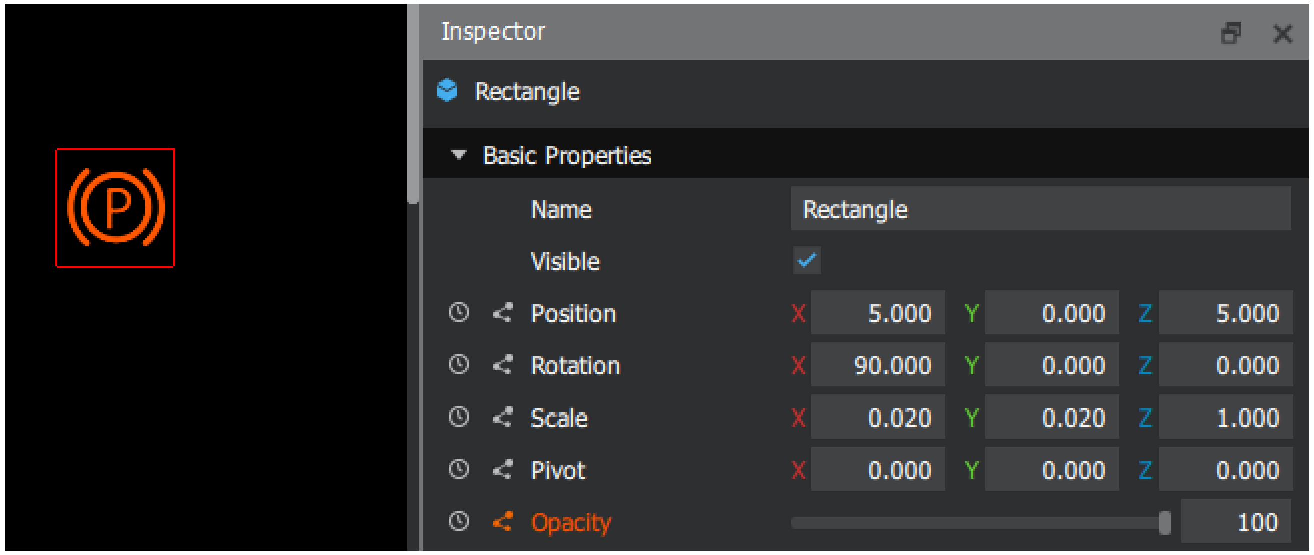
- Final icon color will be set by the diffuse color

How to Create Dashboard Icon

- Create a new rectangle from basic objects menu



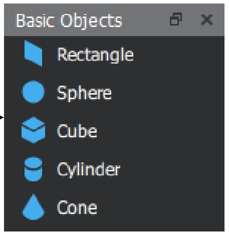
- Adjust position and size of the rectangle and assign the material

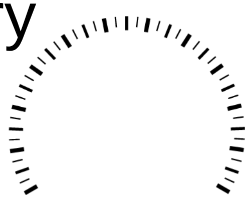


Name	Rectangle					
Visible	✓					
Position	X	5.000	Y	0.000	Z	5.000
Rotation	X	90.000	Y	0.000	Z	0.000
Scale	X	0.020	Y	0.020	Z	1.000
Pivot	X	0.000	Y	0.000	Z	0.000
Opacity	100					

- Icon visibility can be controlled by the opacity attribute

Create Geometry From C++





- It would be really tedious to create complex geometry such as speedometer ticks from basic objects.
  On the other side, geometry created in an external 3D authoring tool (in form of asset) would be hardly adjustable
- One solution is to create such geometry directly in the code
- First, we need to define a structure of our geometry vertices

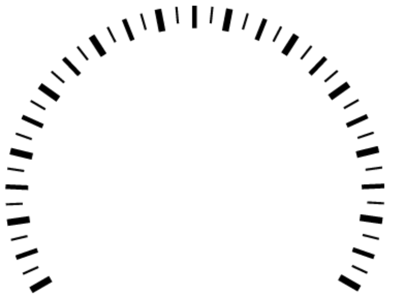
```
struct Vertex
{
  QVector3D position; // vertex position
  QVector3D normal;   // vertex normal (necessary for correct lighting)
  QVector2D uv;       // vertex texture coordinate (necessary for texturing)
};
```

Create Geometry From C++

- Second, we need to set the position (and other attributes) of vertices in our geometry (triangular mesh) in vertex buffer
- The following example will populate the vertex buffer with vertices of a single triangle

```
QVector<Vertex> vertices = { Vertex(5,5,0), Vertex(0,0,0), Vertex(5,0,0) };
QByteArray vertexBuffer( reinterpret_cast<const char *>(vertices.constData()),
  vertices.size() * sizeof( Vertex ) );
```

You need to replace this by computed positions of all vertices of speedometer ticks. To do so, you need to calculate positions of all rectangular ticks to get something like this



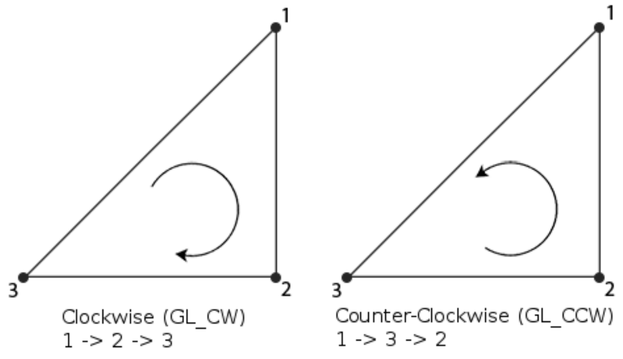
Create Geometry From C++

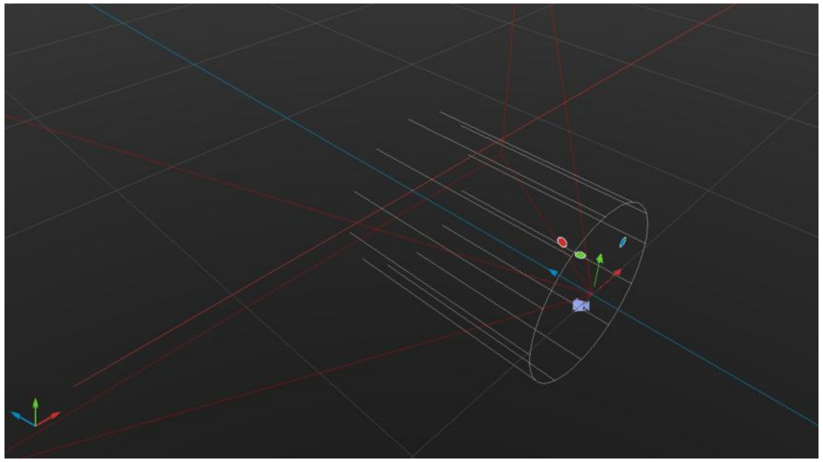
- Note on the order of vertices in a triangle: The front face of the triangle will be determined by the order of the vertices. Triangles are then discarded based on their apparent facing in a process known as face culling.
- This triangle will be **visible** from the camera's point of view

```
QVector<Vertex> vertices = { Vertex(5,5,0), Vertex(0,0,0), Vertex(5,0,0) };
```

- This triangle will be **invisible** from the camera's point of view

```
QVector<Vertex> vertices = { Vertex(5,5,0), Vertex(5,0,0), Vertex(0,0,0) };
```

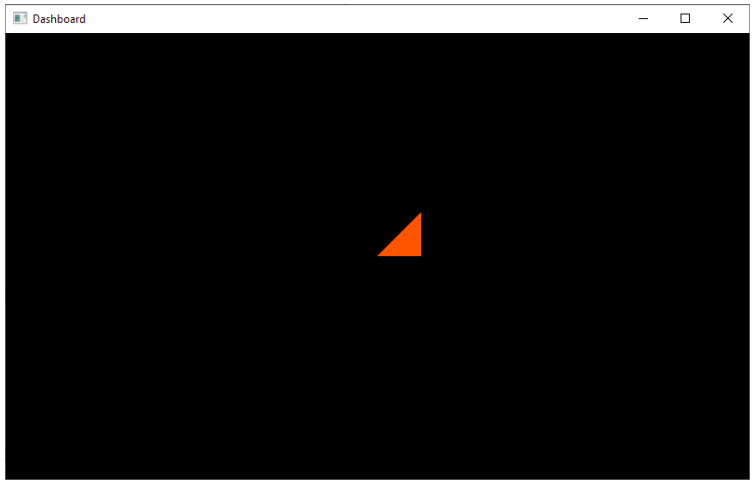




Create Geometry From C++

- Note on the normal of vertices in a triangle: Normal plays a crucial role in shading. Our test triangle should have the normal pointing in the z-direction to be properly illuminated.

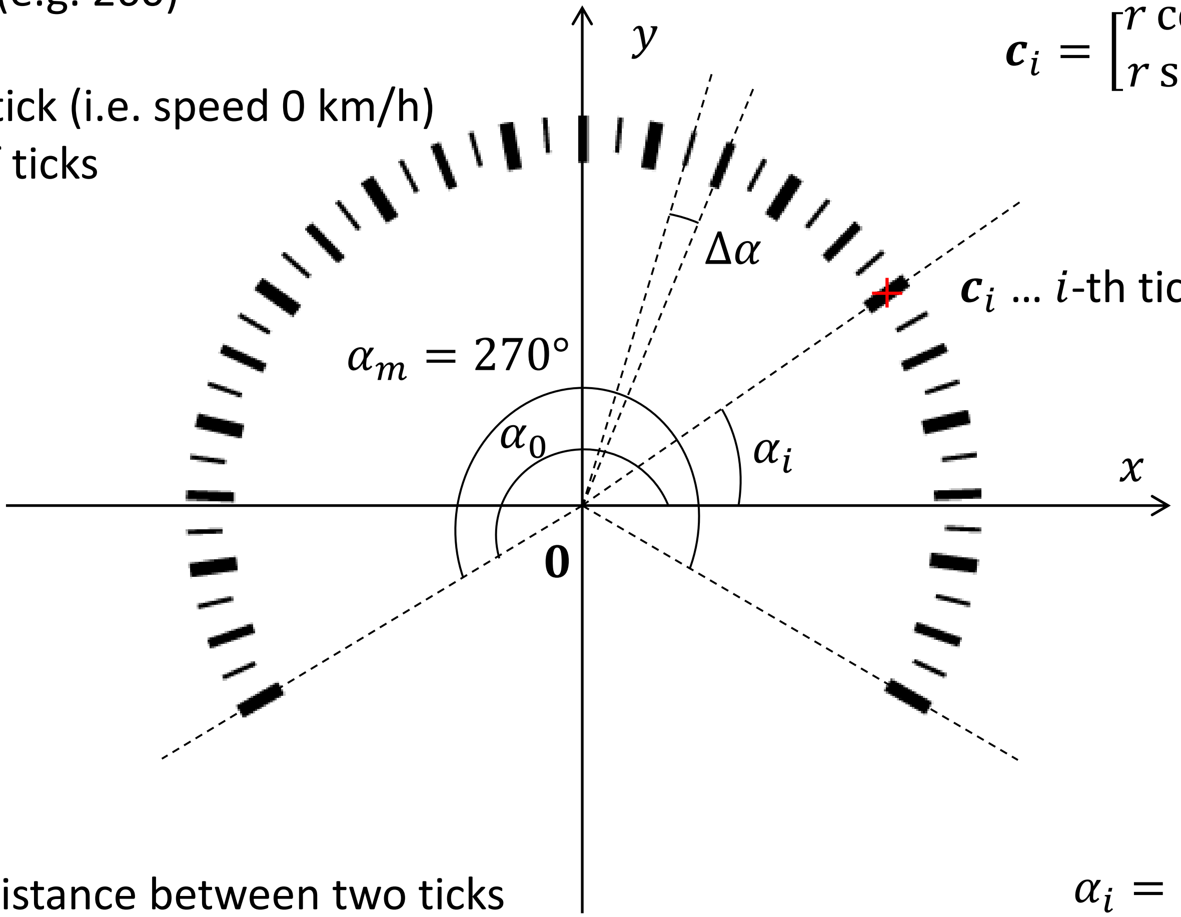
```
struct Vertex
{
  QVector3D position; // vertex position
  QVector3D normal; // vertex normal (necessary for correct lighting)
  QVector2D uv; // vertex texture coordinate (necessary for texturing)

  Vertex( const float x, const float y, const float z ) {
    position.setX( x );
    position.setY( y );
    position.setZ( z );

    normal.setX( 0.0f );
    normal.setY( 0.0f );
    normal.setZ( 1.0f );
  }
};
```



Create Geometry From C++

n … a number of ticks (e.g. 260)
α_i … angle of i-th tick
α_0… angle of the first tick (i.e. speed 0 km/h)
α_m … angular range of ticks

$$\boldsymbol{c}_i = \begin{bmatrix} r\cos\alpha_i \\ r\sin\alpha_i \end{bmatrix} + \boldsymbol{0}$$



$$\alpha_0 = \frac{\alpha_m}{2} + 90$$

$\Delta\alpha = \frac{\alpha_m}{n}$ … angular distance between two ticks

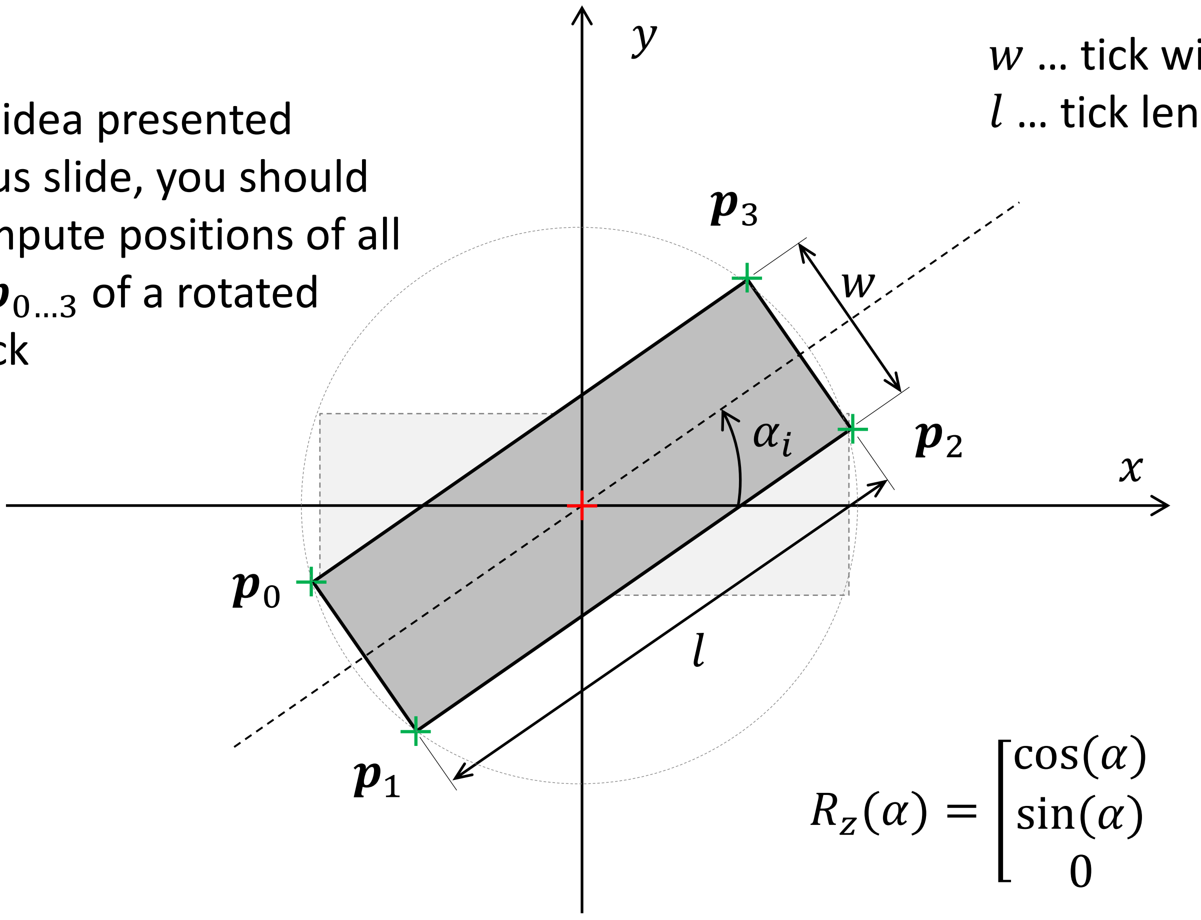
$$\alpha_i = \alpha_0 - i\Delta\alpha$$

Note that all our calculations are done in XY plane. That fact must correspond with the layout of other components in Qt 3D Studio presentation

Create Geometry From C++

Following the idea presented on the previous slide, you should be able to compute positions of all four vertices $\boldsymbol{p}_{0\ldots3}$ of a rotated rectangular tick

w … tick width
l … tick length



$$R_z(\alpha) = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotate point $\boldsymbol{p}_i$ around the origin ($\boldsymbol{0}$) by an angle α: $\boldsymbol{p}_i' = R_z(\alpha)\boldsymbol{p}_i$

Create Geometry From C++

- Third, from the reason we will see later, we have to connect two signals from the presentation to our custom class slots

```
QObject::connect( viewer_.presentation(), SIGNAL( meshesCreated( QStringList,
   QString ) ), this, SLOT( meshesCreated( QStringList, QString ) ) );

QObject::connect( viewer_.presentation(), SIGNAL( elementsCreated( QStringList,
   QString ) ), this, SLOT( elementsCreated( QStringList, QString ) ) );
```

- These signals will inform us about successful creation of meshes and elements in the presentation and we have to react on these situations somehow (on next slides)

A nice explanation of the signals and slots mechanism can be found here
https://doc.qt.io/qt-5/signalsandslots.html

Create Geometry From C++

- Third, we create a new geometry consisting of individual triangles (in this example we deal with one triangle) as follows

```
Q3DSGeometry * geometry = new Q3DSGeometry();
geometry->setPrimitiveType( Q3DSGeometry::PrimitiveType::Triangles );
geometry->setVertexData( vertexBuffer );
geometry->addAttribute( Q3DSGeometry::Attribute::PositionSemantic );
geometry->addAttribute( Q3DSGeometry::Attribute::NormalSemantic );
geometry->addAttribute( Q3DSGeometry::Attribute::TexCoordSemantic );
```

- Now we have to ask the viewer to create a new mesh

```
viewer_.presentation()->createMesh( "SingleTriangle", *geometry );
```

Create Geometry From C++

- Our two slots from the previous slides take place here; the first one is the slot invoked when meshes are created

```
void Dashboard::meshesCreated( const QStringList & meshNames, const QString & error )
{
  qDebug() << meshNames;
  qDebug() << error;

  // we need to create an element for each mesh
  for ( const QString & name : meshNames )
  {
    QHash<QString, QVariant> properties = { { "name", name + "Element" } ,
{ "sourcepath", name },
{ "position", QVector3D( 0.0f, 0.0f, 0.0f ) }, // initial position
{ "scale", QVector3D( 1.0f, 1.0f, 1.0f ) },    // initial scale
{ "rotation", QVector3D( 0.0f, 0.0f, 0.0f ) }, // initial rotation
{ "pivot", QVector3D( 0.0f, 0.0f, 0.0f ) },    // initial center of rotation
{ "material", "Orange"} // material must be already referenced in the scene
    };

    viewer_.presentation()->createElement( "Scene.Layer", "Slide1", properties );
  }
}
```

The list of all attribute names can be found here https://doc.qt.io/qt3dstudio/openglruntime/qt3dstudio-opengl-runtime-attribute-names.html

Default scene paths are used here. This need to be verified in the actual context of the scene

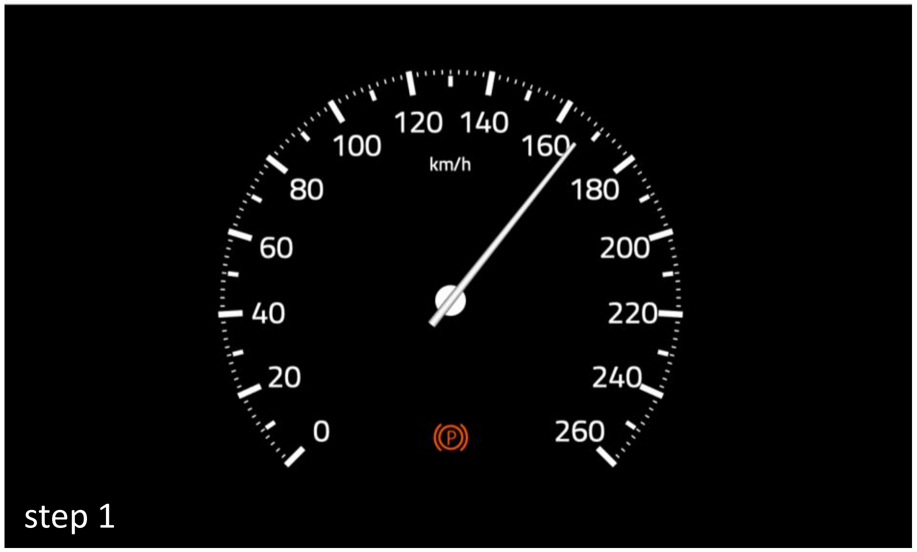
Create Geometry From C++

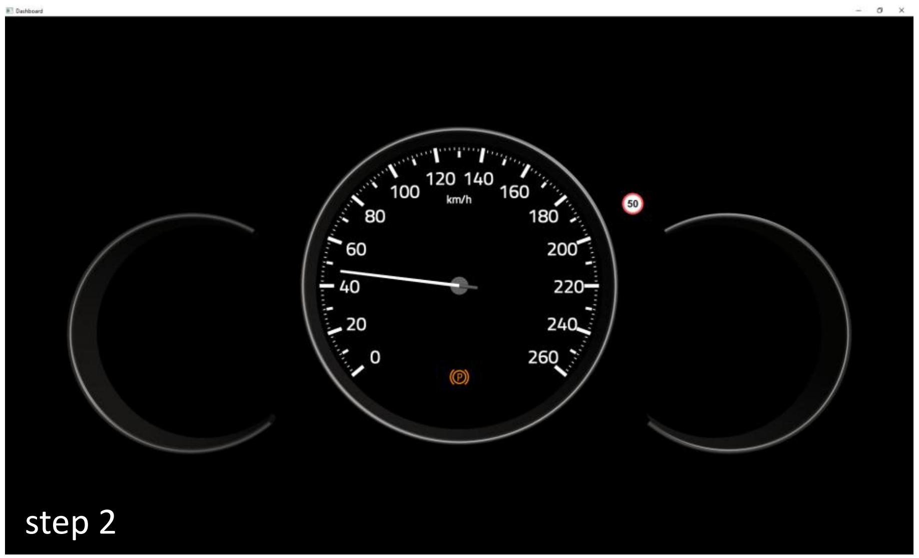
- Our two slots referenced on the previous slides take place here; the second one is invoked when an element is created

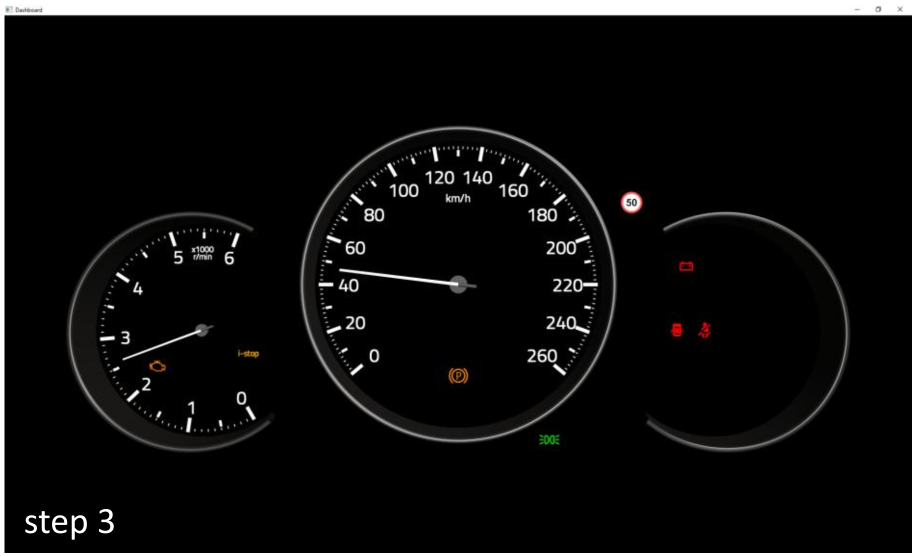
```
void Dashboard::elementsCreated( const QStringList & elementPaths, const QString & error )
{
  qDebug() << elementPaths;
  qDebug() << error;

  // nothing to do here
}
```

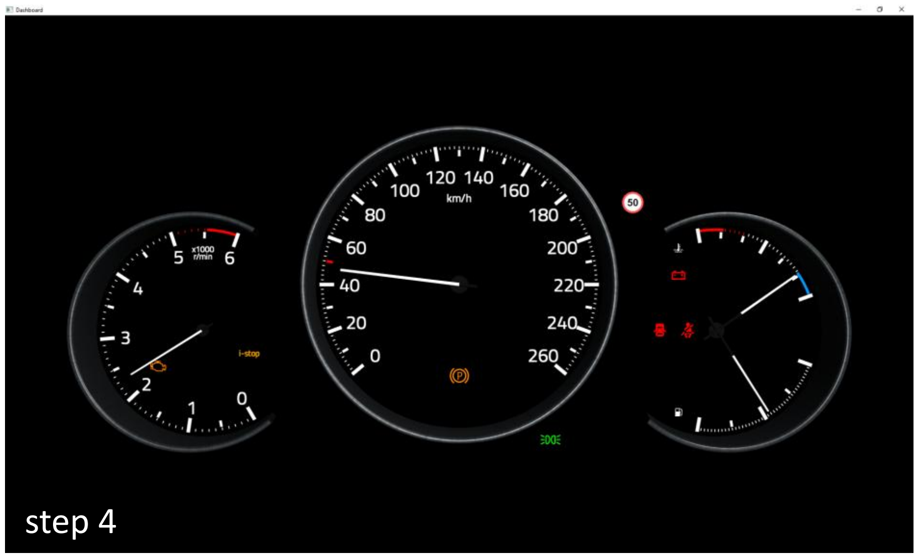
- After the successful invocation of this slot, we should be able to see the newly created geometry

Dashboard in Qt 3DS/C++

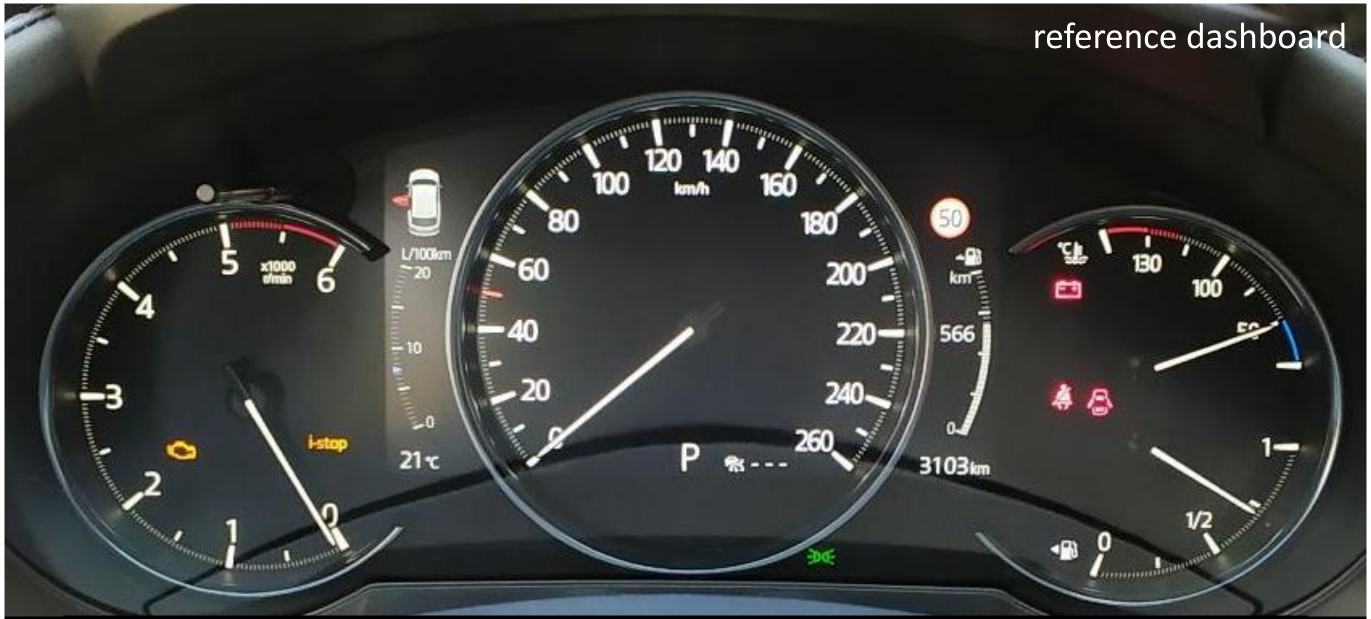




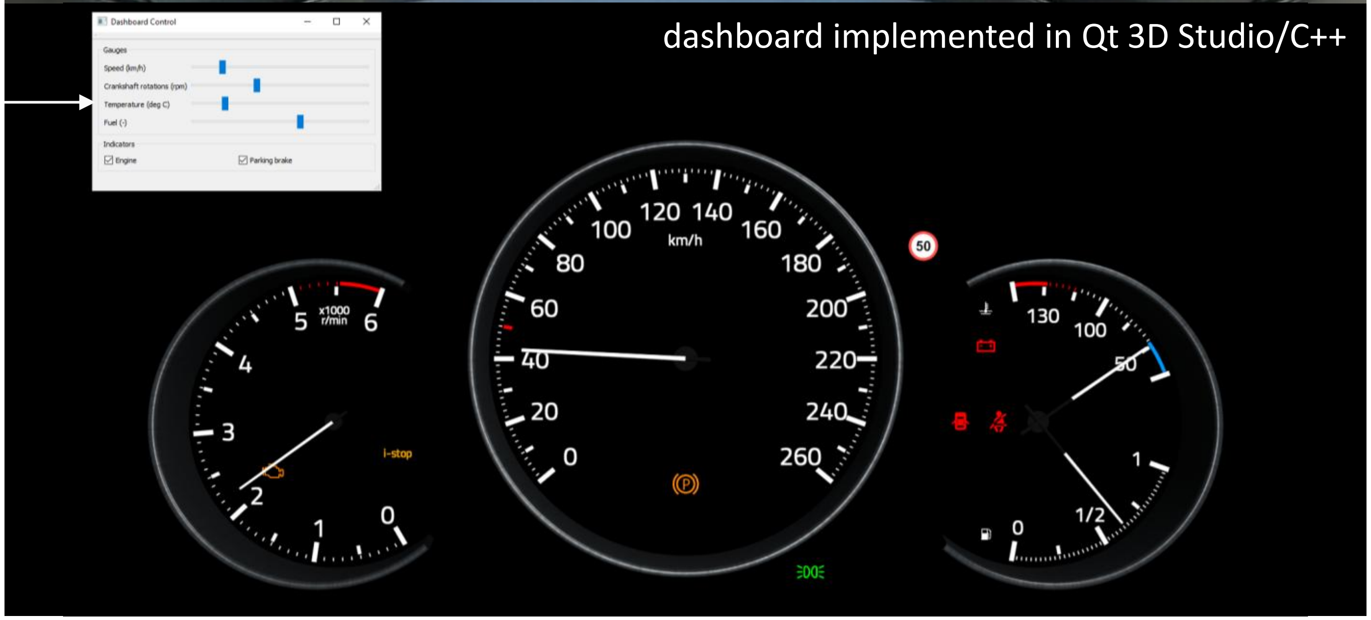




Dashboard in Qt 3DS/C++



The control panel allows to adjust individual dashboard elements



Dashboard Control

Gauges

Speed (km/h)

Crankshaft rotations (rpm)

Temperature (deg C)

Fuel (-)

Indicators

☑ Engine

☑ Parking brake

Other UI (HMI) Design Tools

- Kanzi by Rightware
- Altia 3D
- Crank Storyboard Suite
- Candera CGI Studio
- Kuesa module for Qt 3D
- Qt Safe Renderer (ISO 26262:2018-6, ASIL D)