(1) vývojové prostředí

- qt creator
- qt 5

instalace (viz http://qt-project.org/)

(2) hello world

- vytvoření aplikace z šablony
- její spuštění

(3) qml v qt4 vs qt5

- qmlviever
- qmlscene

-> Jak se to nastaví v Qt Creatoru?
-> Jak nastavit Qt Creator obecně? Kit, Project, Deployment

(4) Minimalistický program

- vytvoření hello world;

(5) Jaká je struktura aplikace

import ... (namespace)
-> sada tříd, které někdo udělal a my je můžeme použít
-> později možná složitější
Rectangle { Rectangle { Rectangle { } } }

// velkými písmeny je jméno třídy (Rectangle), všechno ostatní začíná malými

(6) property

- požívání property (color, width, height)
- vytvoření vlastní property
  -> jaké jsou typy int, string, real, variant
- speciální property id, parent
- signal, onSignal, onClicked, implicitni onPropertyChanged
  -> console.log("x");
  -> Component.onCompleted: {}

(7) javascipt kód

- přístup k property: id_objektu.property = "hodnota";
- je to javascript! Math.round(1.5); parseInt("07", 10);

příklad: Zkuste si vyrobit obdélníky. Nějak je umístit a pracovat s nimi

```
Rectangle {
  Rectangle {
    x: 0; y: 0; width: 10; height: 10;
    color: "red"
  }
  Rectangle {
    x: 10; y: 0; width: 10; height: 10;
    color: "blue"
  }
  Rectangle {
    id: rectC
    x: 0; y: 10; width: 10; height: 10;
    color: "yellow"
  }

  Component.onCompleted: {
    rectC.x = 10;
  }
}
```

(8) Typy základních objektu:

- Retangle {}
- MouseArea {}
- Item {}
- Component {}

(9) Anchors

- jak je použít
- složená property

příklad: využijte k umístění obdélníků z minulého příkladu anchors.

(10) dědičnost

- Button
- (import "podadresare")
- property alias x: textField.text;

příklad: vytvořte si vlastní tlačítko

```
Rectangle {
  id: button;
  width: 40
  height: 120;
  color: "red";
  property alias buttonText: text.text.
  signal clicked();
  Text {
    id: text;
    anchors.centerIn: parent;
    color: "green";
  }
  MouseArea {
    anchors.fill: parent;
    onClicked: {
      button.clicked();
    }
  }
}
```

(11) ListModel, ListView, Delegate

```
ListModel {
  id: mujModel
  ListElement { name: “nana“; abc: “lala“ }
}
ListView{
  model: mujModel
  delegate: Rectangle{
    Text { anchors.centerIn:parent; text:model.name; }
  }
}
```

```
import QtQuick.XmlListModel 2.0

XmlListModel {
    id: xmlModel
    source: "http://www.linuxalt.cz/rss.xml"
    query: "/rss/channel/item"

    XmlRole { name: "title"; query: "title/string()" }
    XmlRole { name: "pubDate"; query: "pubDate/string()" }
}
```

(12) PropertyAnimation

```
    Behavior on width {
        NumberAnimation { duration: 1000 }
    }
```

(13) Ladění aplikace

```
function f() {
  var x = 12
  console.assert(x == 12, "This will pass");
  console.assert(x > 12, "This will fail");
}

function f() {
    console.time("wholeFunction");
    console.time("firstPart");
    // first part
    console.timeEnd("firstPart");
    // second part
    console.timeEnd("wholeFunction");
}

function f() {
  console.count("f called");
}

function f() {
    console.profile();
    //Call some function that needs to be profiled.
    //Ensure that a client is attached before ending
    //the profiling session.
    console.profileEnd();
}
```

(14) aplikace jako „exe“

```
#include <QtGui/QGuiApplication>
#include <QQmlApplicationEngine>
#include <QQuickWindow>
#include "qtquick2applicationviewer.h"

int main(int argc, char *argv[])
{

   QGuiApplication app(argc, argv);
   QQmlApplicationEngine engine;

   engine.load(QUrl("qml/igcTest2/main.qml"));
   QObject *topLevel = engine.rootObjects().value(0);
   QQuickWindow *window = qobject_cast<QQuickWindow *>(topLevel);
```

```
    window->show();
    return app.exec();

}
```

(15) C++ bidning

```
 // MyItem.qml
 import QtQuick 1.0

 Text { text: currentDateTime }

 QDeclarativeView view;
 view.rootContext()->setContextProperty("currentDateTime",
QDateTime::currentDateTime());
 view.setSource(QUrl::fromLocalFile("MyItem.qml"));
 view.show();


 qmlRegisterType<ImageViewer>("MyLibrary", 1, 0, "ImageViewer");
 import MyLibrary 1.0

 ImageViewer { image: "smile.png" }



 class ImageViewer : public QDeclarativeItem

 {
     Q_OBJECT
     Q_PROPERTY(QUrl image READ image WRITE setImage NOTIFY imageChanged)

 public:
     void setImage(const QUrl &url);
     QUrl image() const;

 signals:
     void imageChanged();
 };
```

Reference:

http://qt-project.org/doc/qt-5.0/qtquick/qtquick-debugging.html

http://qt-project.org/doc/