Współczesne Narzędzia Obliczeniowe
Laboratorium II

Zasady ogólne

Przedmiot składa się z 15 zajęć komputerowych. Na każdych zajęciach można zdobyć od 0 do 5 punktów za zadania standardowe (punkty są typu *integer*), zadania opcjonalne są premiowane dodatkowo oznaczone jako (⋆x pkt). W niektórych zagadnieniach mogą wystąpić punkty ujemne. Co oznacza że można standardowo zdobyć **(75 pkt)**. Prawie każde zadanie (pomijając ostatnie) można donieść najpóźniej tydzień po jego oficjalnym terminie za połowę punktów (zaokrąglając w dół). Próg zaliczenia to: **(46 pkt)** – średnia > 3.0. Na laboratorium **nie wolno** używać gotowych skryptów (dotyczy to zarówno skryptów pobranych z internetu, jak i *pożyczonych* od kolegów). Użycie gotowych skryptów zeruje punktację z danego laboratorium, bez możliwości poprawy. Punktacja za skrypty zbyt podobne do siebie dzielona jest na liczbę osób piszących wspólnie. Laboratorium nie jest poprawiane. Nieobecności należy zgłosić przed zajęciami, zaś zadanie odrobić na równoległych laboratoriach lub donieść po tygodniu za połowę punktów. Materiały na zajęcia można znaleźć pod linkiem: `http://bit.ly/2Sgq1dA`.

1. Machine Learning

1.1. OpenCV

Zagadnienie poruszane na laboratorium polega na przetwarzaniu wstępnym danych wizualnych potrzebnych do wytrenowania sieci neuronowych. Na początek, należy pobrać i rozpakować odpowiednie archiwum z danymi. Następnie z poziomu skryptu przy użyciu modułów `import os, sys, glob` należy przeskanować wskazany w wierszu poleceń (domyślnie domowy) folder i wczytać znajdujące się tam obrazy **(1 pkt)**. Na każdym z obrazów zastosować serię operacji z użyciem modułu `cv2` aby uzyskać usunięcie tła **(1 pkt)**, obrót tak, aby długa krawędź obiektu na obrazie była pozioma **(1 pkt)**, zaś krótsza znajdowała się po lewej stronie **(1 pkt)**. Odpowiednio przyciąć i przeskalować wszystkie obrazy **(1 pkt)**.

1.2. Tenforflow – EMNIST

Celem laboratorium jest zapoznanie się z podstawowym framework'iem tensorflow. W tym celu należy go zainstalować z poziomu wiersza poleceń poprzez komendę `pip install tensorflow tensorboard`. Uwaga – z reguły tensorflow nie jest kompatybilny z najnowszą wersją pythona/winpythona. Po instalacji należy zapoznać się z tutorialem dotyczącym datasetu MNIST **(1 pkt)**. Główne zadanie polega na dostosowaniu skryptu tak aby sieć rozpoznawała określone znaki z bazy EMNIST (dane określone przez prowadzącego). Bazę standardowo należy podzielić na część uczącą, testową i weryfikacyjną. Skuteczność rozpoznawania części weryfikacyjnej przekraczająca 50% **(1 pkt)**, 75% **(1 pkt)**. Wizualizacja procesu uczenia za pomocą tensorboard **(1 pkt)**. Przetestowanie skryptu na zewnętrznym obrazku podanym z wiersza poleceń **(1 pkt)**. Eksport danych uczących do pliku tfrecord (⋆1 pkt). Eksport sieci do h5 i json (⋆1 pkt).

1.3. *Tenforflow – images*

Celem laboratorium jest stworzenie sieci neuronowej wykrywającej określone narzędzie na obrazie. Aby wykonać zadanie należy wygenerować 200 różnych obrazków na podstawie dowolnych 10 obrazów pobranych z sieci i wstawionych w nie w **losowe miejsca** i pod **losowym kątem** spreparowanych we wcześniejszej laborce narzędzi **(2 pkt)**. Następnie stworzyć i wytrenować sieć (może być za pomocą framework'a Keras) która osiągnie skuteczność powyżej 75% **(2 pkt)**. Dodatkowo do danych treningowych dla podniesienia skuteczności można dołączyć dane z imagenet (plik z linkami udostępniony na dysku). Quasi-skuteczne przetestowanie na jednym **losowym** obrazie z imagenet **(1 pkt)**.

1.4. *OpenAi Gym*

Zainstalować bibliotekę (o ile nie jest zainstalowana) OpenAI Gym `pip install gym`. Należy wykonać zagadnienie sterowania klasycznego i poprzez uczenie ze wzmocnieniem w środowisku wskazanym przez prowadzącego. Za wykonanie sterowania za pomocą klawiszy wczytywanych z klawiatury **(1 pkt)**. Użycie klasycznych metod aby wysterować obiekt – standardowe sterowniki, SVC lub inne **(2 pkt)**. Użycie uczenia ze wzmocnieniem w celu wysterowania obiektu **(2 pkt)**.

Dodatkowo w przypadku użycia dowolnego środowiska graficznego z serii Gym Retro (SEGA lub Atari2600) i zastosowaniu w problemie uczenia ze wzmocnieniem na podstawie danych wizyjnych można uzyskać **(⋆5 pkt)**. **Dodatkową** część zadania to można **wyjątkowo** donieść na następne zajęcia **bez utraty punktów**.

2. *Object Orienting Programming*

Seria 6 zajęć laboratoryjnych mająca na celu implementację wybranego środowiska wieloagentowego z agentami autonomicznymi. Wymagane jest użycie klas.

2.1. *Console*

Laboratorium ma na celu implementację uproszczonej wersji środowiska w konsoli. Przygotowanie podstawowej planszy przy użyciu klas i wyrysowanie jej na konsoli **(1 pkt)**. Użycie kolorów **(⋆1 pkt)**. Brak klas określających pole oraz agentów skutkuje **(−2 pkt)**. Implementacja reguł środowiska i mechaniki poruszania się agentów **(3 pkt)**. Implementacja możliwości poruszania się agentem za pomocą poleceń bądź klawiszy **(1 pkt)** wczytanych z zakończeniem linii – standardowym `input()`. Z kolei jeśli uniknie się zakończeń linii – czyli jak za pomocą typowego `getch()` **(⋆1 pkt)**.

2.2. *PyQt – simple GUI*

Zadanie realizowane na laboratorium polega na implementacji prostego GUI w za pomocą biblioteki PyQt. Obsługa przycisków start, nowa gra, wyście **(1 pkt)**. Poruszanie się agentem za pomocą przycisków **(1 pkt)**, blokowanie ruchu przeciwników w trakcie poruszania się agentem **(1 pkt)**. Automatyczne odświeżanie planszy **(1 pkt)**. Użycie prostych grafik lub kolorów przy wyświetlaniu planszy **(1 pkt)**. Jednoczesne użycie wyświetlania planszy na kontrolkach Qt i za pomocą tekstu na kontrolce `QTextEdit` **(⋆2 pkt)**. Brak odpowiednio użytych klas skutkuje **(−2 pkt)**.

2.3. *PyQt – QGraphicsScene, QGraphicsItem +1w*

Przestawienie się z prostych kontrolek na zaawansowane wraz z renderowaniem agentów i środowiska jest zagadnieniem niniejszej laborki. Renderowanie sceny z użyciem `QGraphicsScene` **(2 pkt)**. Użycie aktywnych agentów dziedziczących po `QGraphisItem` i samorysujących się **(2 pkt)**. Wyświetlanie grafik zewnętrznych (zasoby) za pomocą `QGraphisItem` **(1 pkt)**. Zadanie można donieść tydzień później bez utraty punktów.

2.4. *PyQt – Socket*

Zadanie polega na opracowaniu i zaimplementowaniu działania w środowisku wielu agentów łączących się za pomocą TCP/IP. Implementacja protokołu **(1 pkt)**. Pełne działanie **(4 pkt)**.

2.5. History export/import – json/xml

Zapisanie i wczytanie pojedynczego stanu środowiska oraz agentów do postaci pliku XML **(2 pkt)**. Opracowanie systemu odtwarzania wszystkich stanów z XML **(3 pkt)**. Zapisanie i wczytywanie parametrów początkowych z pliku JSON **(⋆2 pkt)**.

2.6. AI

Zadanie polega na implementacji bota. Przetrwanie przez bota minuty w środowisku **(1 pkt)**, użycie losowych akcji **(1 pkt)**, wygrana z człowiekiem **(3 pkt)**.

3. Python and scripts

3.1. SPAM Filter

Napisać skrypt w Pythonie który przeskanuje folder w poszukiwaniu plików tekstowych zawierających maile, wczytać je i sparsować je do postaci listy instancji pewnych **klas**. Klasa powinna mieć pola: nadawca, odbiorca, data, tytuł i treść, oraz metody: wyświetlającą maila i wczytującą maila z pliku **(1 pkt)**. Umożliwić wystąpienia polskich znaków, konwertując kodowanie tekstu na ASCII **(1 pkt)**. Stworzyć **słowniki** (zmienne słownikowe, wewnątrz programu) spamu i hamu na podstawie liczby wystąpień danego słowa w mailach i policzyć prawdopodobieństwa warunkowe słów $P(word_i|SPAM)$ i $P(word_i|HAM)$. Przetestować wykrywanie spamu na pliku example.txt – jakie jest prawdopodobieństwo, że dana wiadomość jest spamem **(1 pkt)**. Zmienić sposób obliczania prawdopodobieństwa - wygładzanie Lapace'a $k = 2$, $possible_{types} = 2$ (SPAM $\vee$ HAM) **(1 pkt)**, połączenie słowniki z dict.xml oraz stworzony w ramach zadania **(1 pkt)**. Skonstruować inteligentny słownik z użyciem odmiany wyrazów **(⋆1-3 pkt)**.
Trochę prawdopodobieństwa przy założeniu zdarzeń niezależnych:

$$P(SPAM) = \frac{count(massages\ in\ SPAM)}{count(massages)} \tag{1}$$

Wygładzanie Laplace'a

$$P(SPAM) = \frac{count(massages\ in\ SPAM) + k}{count(massages) + k * possible_{types}} \tag{2}$$

$$P(word_i|SPAM) = \frac{count(word_i\ in\ SPAM)}{count(words\ in\ SPAM) + k * possible_{types})} \tag{3}$$

$$P(massage|SPAM) = \prod_i P(word_i|SPAM) \tag{4}$$

$$P(SPAM|massage) = \frac{P(massage|SPAM) * P(SPAM)}{P(massage|SPAM) * P(SPAM) + P(massage|HAM) * P(HAM)} \tag{5}$$

3.2. TCP/IP

Napisać skrypt w Pythonie mający na celu stworzenie komunikatora internetowego. Użycie odpowiednich klas **(1 pkt)**. Komunikacja synchroniczna **(2 pkt)**. Komunikacja asynchroniczna z użyciem wątków **(2 pkt)**. Przesłanie za pomocą TCP/IP dokumentu *.doc **(⋆2 pkt)**.

3.3. SMTP

Należy napisać skrypt potrafiący obsługiwać skrzynkę pocztową (SMTP). Odczyt maili **(1 pkt)**, wysyłanie maili **(1 pkt)**. Wysłanie załącznika w postaci obrazu **(1 pkt)**, jego odbiór i zapis **(2 pkt)**.

3.4. Tetris +1w

Należy rozwiązać zagadnienie paletyzacji. Na planszy 15x25 należy losowo umieścić 15 klocków (o 3 różnych stylach) nienachodzących na siebie, posiadających zdefiniowaną adresację **(1 pkt)**. Poruszanie klockami za pomocą klawiszy **(2 pkt)**, przy założeniu że klocki nie mogą na siebie nachodzić. Głównym celem jest opracowanie algorytmu który stawi wszystkie klocki w jednym rogu minimalizując zajmowaną przez nie przestrzeń **(2 pkt)**. Zagadnienie można donieść po tygodniu bez utraty punktów.

3.5. Regex

Napisać skrypt w Pythonie który pobierze wskazany plik tekstowy zawierający fragment tekstu łacińskiego z pomieszanymi adresami mailowymi. Za pomocą pakietu *re* należy znaleźć w tekście poprawne adresy mailowe. Znalezienie **poprawnych** 3 adresów – **(3 pkt)**, 6 – **(4 pkt)**, 9 – **(5 pkt)**. Każdy błędnie znaleziony adres **(−1 pkt)** punkt. Upakowanie wyszukiwania w jedno zapytanie regularne **(⋆1 pkt)**.
Zasady poprawnego nazywania adresów mailowych:

- niceandsimple@example.com
- very.common@example.com
- a.little.lengthy.but.fine@dept.example.com
- disposable.style.email.with+symbol@example.com
- user@[IPv6:2001:db8:1ff::a0b:dbd0]
- ”much.more unusual”@example.com
- ”very.unusual.@.unusual.com”@example.com
- `"very.(),:;<>[]\".VERY.\"very@\\ \"very\".unusual"@strange.example.com`
- postbox@com (domeny najwyższego rzędu)
- admin@mailserver1 (nazwa domeny lokalnej)
- `!#$%&'*+-/=?^_`{}|~@example.org`
- `"()<>[]:,;@\\\"!#$%&'*+-/=?^_`{}| ~.a"@example.org`
- ” ”@example.org (spacja w uszach)

Niewłaściwe adresy mailowe:

- Abc.example.com (musi być @)
- A@b@c@example.com (ale tylko jedna)
- `a"b(c)d,e:f;g<h>i[j\k]l@example.com`
- just”not”right@example.com (uszy tam gdzie trzeba)
- `this is"not\allowed@example.com`
- `this\ still\"not\\allowed@example.com`

Zwalidowane adresy: soldat@eu.org; mauris@wp.pl; ”erat nulla”@example.com; Fusce+felis.enim.viverra@com; Suspendisse+eu.est@rop.eu; \tincidunt”@com.dot; ” ”@null.nul; ”erat@nec”@gmail.com; luctus@[IPv6:2001:db8::ff00:42:8329].

Z kolei do złych adresów należą: sem.lectus@org; hac@eta; ac@rutrum@non.qor; nunc”in dui@.org.pl; Sed.tincidunt”,”dolor@tada.tede; non@volutpat.

Przykłady i podpowiedzi

1. Image processing

Przykłady poniżej zaczerpnięte zostały ze strony https://opencv-python-tutroals.readthedocs.io/en/latest/index.html. Dotyczą binearyzacji obrazu, znajdowania otoczki wypukłej oraz wykrywania linii za pomocą transformaty Hough.

Treshold

```
mg = cv2.imread('gradient.png',0)
ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)

titles = ['Original Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]

for i in xrange(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])

plt.show()
```

Convex Hull

```
mg = cv2.imread('star.jpg')
img_gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(img_gray, 127, 255,0)
contours,hierarchy = cv2.findContours(thresh,2,1)
cnt = contours[0]

hull = cv2.convexHull(cnt,returnPoints = False)
defects = cv2.convexityDefects(cnt,hull)

for i in range(defects.shape[0]):
    s,e,f,d = defects[i,0]
    start = tuple(cnt[s][0])
    end = tuple(cnt[e][0])
    far = tuple(cnt[f][0])
    cv2.line(img,start,end,[0,255,0],2)
    cv2.circle(img,far,5,[0,0,255],-1)

cv2.imshow('img',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Hough transform

```
mg = cv2.imread('dave.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray,50,150,apertureSize = 3)

lines = cv2.HoughLines(edges,1,np.pi/180,200)
for rho,theta in lines[0]:
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))

    cv2.line(img,(x1,y1),(x2,y2),(0,0,255),2)

cv2.imwrite('houghlines3.jpg',img)
```

2. *Tenforflow*

Przykład tworzenia sieci neuronowej za pomocą tensorflow.

Create TF net

```
ef createAndTrainTF(inputs, desired_outputs, testIn, labels, colors):
    INPUT_FEATURES_LEN = len(inputs[0])
    HIDDEN_UNITS_L1 = 64
    HIDDEN_UNITS_L2 = 32
    OUTPUT_LEN = len(desired_outputs[0])
    SIZE = len(inputs)
    tr = int((0.8*SIZE))
    val = tr + int((0.1*SIZE))
    tf.reset_default_graph()
    inputPlaceholder = tf.placeholder(tf.float32, [None, INPUT_FEATURES_LEN])
    outputPlaceholder = tf.placeholder(tf.float32, [None, OUTPUT_LEN])

    X_tr = inputs[:tr]
    X_val = inputs[tr:val]
    X_te = inputs[val:]

    y_tr = desired_outputs[:tr]
    y_val = desired_outputs[tr:val]
    y_te = desired_outputs[val:]

    # layers:
    weights_1 = tf.Variable(tf.truncated_normal([INPUT_FEATURES_LEN, HIDDEN_UNITS_L1]))
    biases_1 = tf.Variable(tf.zeros([HIDDEN_UNITS_L1]))
    layer_1_outputs = tf.nn.relu(tf.matmul(inputPlaceholder, weights_1) + biases_1)
    weights_2 = tf.Variable(tf.truncated_normal([HIDDEN_UNITS_L1, HIDDEN_UNITS_L2]))
    biases_2 = tf.Variable(tf.zeros([HIDDEN_UNITS_L2]))
    layer_2_outputs = tf.nn.relu(tf.matmul(layer_1_outputs, weights_2) + biases_2)
    # output layer
    weights_3 = tf.Variable(tf.truncated_normal([HIDDEN_UNITS_L2, OUTPUT_LEN]))
    biases_3 = tf.Variable(tf.zeros([OUTPUT_LEN]))
    logits = tf.nn.softmax(tf.matmul(layer_2_outputs, weights_3) + biases_3)

    error_function = tf.div(tf.reduce_sum((outputPlaceholder-logits)**2), OUTPUT_LEN)
    optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
    grads_and_vars = optimizer.compute_gradients(error_function)
    train_op = optimizer.apply_gradients(grads_and_vars)
    correct_prediction = tf.div(tf.reduce_sum(outputPlaceholder-logits), OUTPUT_LEN*batchSize)

    epochs = 15000
    train_cost, val_cost, val_acc = [], [], []
    gpuOpts = tf.GPUOptions(per_process_gpu_memory_fraction=0.2)
```

Przykład uczenia i ewaluacji

Train TF

```
ith tf.Session(config=tf.ConfigProto(gpu_options=gpuOpts)) as sess:
        # Saving tf board
        merged = tf.summary.merge_all()
        writer = tf.summary.FileWriter("logs", sess.graph)
        # initializing all variables
        init = tf.global_variables_initializer()
        sess.run(init)

        for e in range(epochs):
            # TRAINING
            batch_x, batch_y = next_batch(batchSize, X_tr, y_tr)
            feed_dict_train = {inputPlaceholder: batch_x, outputPlaceholder: batch_y}
            fetches_train = [train_op, error_function]
            res = sess.run(fetches=fetches_train, feed_dict=feed_dict_train)
            train_cost += [res[1]]

            # VALIDATING
            feed_dict_valid = {inputPlaceholder: X_val, outputPlaceholder: y_val}
            fetches_valid = [error_function, correct_prediction]
            res = sess.run(fetches=fetches_valid, feed_dict=feed_dict_valid)
            val_cost += [res[0]]
            val_acc += [res[1]]

            if e % 10 == 0:
                print("Epoch %i, Train Cost: %0.3f\tVal Cost: %0.3f\t Val acc: %0.3f" % (
                    e, train_cost[-1], val_cost[-1], val_acc[-1]))

        # TESTING
        feed_dict_test = {inputPlaceholder: X_te, outputPlaceholder: y_te}
        fetches_test = [error_function, correct_prediction]

        # running the validation
        res = sess.run(fetches=fetches_test, feed_dict=feed_dict_test)
        test_cost = res[0]
        test_acc = res[1]
        print("\nTest Cost: %0.3f\tTest Accuracy: %0.3f" % (test_cost, test_acc))

        classes = logits.eval(feed_dict={inputPlaceholder: testIn})
```

3. *OpenAi Gym*

Gym

```
mport gym
env = gym.make('CartPole-v0')
for i_episode in range(20):
    observation = env.reset()
    for t in range(100):
        env.render()
        print(observation)
        action = env.action_space.sample()
        observation, reward, done, info = env.step(action)
        if done:
            print("Episode finished after {} timesteps".format(t+1))
            break
```

4. *PyQt*

Przykładowy kod PyQt w wersji 5 ze strony http://zetcode.com/gui/pyqt5/firstprograms/.

PyQt

```
mport sys
from PyQt5.QtWidgets import QWidget, QPushButton, QApplication

class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        qbtn = QPushButton('Quit', self)
        qbtn.clicked.connect(QApplication.instance().quit)
        qbtn.resize(qbtn.sizeHint())
        qbtn.move(50, 50)

        self.setGeometry(300, 300, 250, 150)
        self.setWindowTitle('Quit button')
        self.show()


if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

5. *QGraphicsScene*

Użycie QGraphicsScene i QGrapicsItem za pomocą przykładowych kodów pobranych z https://github.com/mmisono/pyqt5-example/blob/master/tic_tac_toe.py.

QGraphicsScene – Main

```
!/usr/bin/env python

from PyQt5.QtCore import (QLineF, QPointF, QRectF, Qt)
from PyQt5.QtGui import (QBrush, QColor, QPainter)
from PyQt5.QtWidgets import (QApplication, QGraphicsView, QGraphicsScene, QGraphicsItem,
                             QGridLayout, QVBoxLayout, QHBoxLayout,
                             QLabel, QLineEdit, QPushButton)
if __name__ == '__main__':
    import sys
    app = QApplication(sys.argv)
    mainWindow = MainWindow()

    mainWindow.show()
    sys.exit(app.exec_())
```

QGraphicsItem

```
lass TicTacToe(QGraphicsItem):
    def __init__(self):
        super(TicTacToe, self).__init__()
        self.board = [[-1, -1, -1],[-1, -1, -1], [-1, -1, -1]]
        self.O = 0
        self.X = 1
        self.turn = self.O

    def reset(self):
        for y in range(3):
            for x in range(3):
                self.board[y][x] = -1
        self.turn = self.O
        self.update()

    def select(self, x, y):
        if x < 0 or y < 0 or x >= 3 or y >= 3:
            return
        if self.board[y][x] == -1:
            self.board[y][x] = self.turn
            self.turn = 1 - self.turn

    def paint(self, painter, option, widget):
        painter.setPen(Qt.black)
        painter.drawLine(0,100,300,100)
        painter.drawLine(0,200,300,200)
        painter.drawLine(100,0,100,300)
        painter.drawLine(200,0,200,300)

        for y in range(3):
            for x in range(3):
                if self.board[y][x] == self.O:
                    painter.setPen(Qt.red)
                    painter.drawEllipse(QPointF(50+x*100, 50+y*100), 30, 30)
                elif self.board[y][x] == self.X:
                    painter.setPen(Qt.blue)
                    painter.drawLine(20+x*100, 20+y*100, 80+x*100, 80+y*100)
                    painter.drawLine(20+x*100, 80+y*100, 80+x*100, 20+y*100)

    def boundingRect(self):
        return QRectF(0,0,300,300)

    def mousePressEvent(self, event):
        pos = event.pos()
        self.select(int(pos.x()/100), int(pos.y()/100))
        self.update()
        super(TicTacToe, self).mousePressEvent(event)
```

QGraphicsView

```
lass MainWindow(QGraphicsView):
    def __init__(self):
        super(MainWindow, self).__init__()
        scene = QGraphicsScene(self)
        self.tic_tac_toe = TicTacToe()
        scene.addItem(self.tic_tac_toe)
        scene.setSceneRect(0, 0, 300, 300)
        self.setScene(scene)
        self.setCacheMode(QGraphicsView.CacheBackground)
        self.setWindowTitle("Tic Tac Toe")

    def keyPressEvent(self, event):
        key = event.key()
        if key == Qt.Key_R:
            self.tic_tac_toe.reset()
        super(MainWindow, self).keyPressEvent(event)
```

6. *json*

Dumps

```
mport json

x = {
  "name": "John",
  "age": 30,
  "married": True,
  "divorced": False,
  "children": ("Ann","Billy"),
  "pets": None,
  "cars": [
    {"model": "BMW 230", "mpg": 27.5},
    {"model": "Ford Edge", "mpg": 24.1}
  ]
}

print(json.dumps(x))
```

Read json

```
mport json

# some JSON:
x =  '{ "name":"John", "age":30, "city":"New York"}'

# parse x:
y = json.loads(x)

# the result is a Python dictionary:
print(y["age"])
```

7. *xml*

Minidom jest opisany na stronie: https://wiki.python.org/moin/MiniDom.

8. Other

Dicts

```
>> knights = {'gallahad': 0.9, 'robin': 0.000005, 'lancelot' : 0.1}
>>> knights
{'gallahad': 0.9, 'robin': 5e-06, 'lancelot': 0.1}

>>> knights = dict(gallahad=0.9, robin=0.000005, lancelot=0.1)   # konstruktor
>>> knights
{'gallahad': 0.9, 'robin': 5e-06, 'lancelot': 0.1}

>>> knights['robin']
5e-06
>>> knights['abraham'] = 0.0000000001   # dodawanie elementu
>>> knights
{'gallahad': 0.9, 'abraham': 1e-10, 'robin': 5e-06, 'lancelot': 0.1}

>>> del knights['abraham']    # usuwanie elementu
>>> knights
{'gallahad': 0.9, 'robin': 5e-06, 'lancelot': 0.1}

>>> knights['robin'] = 0.2
>>> knights
{'gallahad': 0.9, 'robin': 0.2, 'lancelot': 0.1}

>>> for k, v in knights.iteritems():    # iterkeys() -tylko po kluczach
...     print k, v         # itervalues() -tylko po wartosciach
gallahad 0.9
robin 0.2
lancelot 0.1

>>> import os
>>> os.listdir('.')       # klasyczny dir

>>> import glob
>>> glob.glob('./*.txt')    # filtrowanie listy plikow

>>> import codecs      # otwieranie plikow z innym kodowaniem
>>> f = codecs.open('file','r','UTF-8')
```

TCP/IP Server

```
mport socket
import sys

# Create a TCP/IP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Tutaj podajemy adres IP własnej maszyny i port
server_address = ('localhost', 10000)
print('starting up on {} port {}'.format(*server_address))
sock.bind(server_address)

# Nasłuchiwanie
sock.listen(1)

while True:
    # Oczekiwanie na połączenie
    print('waiting for a connection')
    connection, client_address = sock.accept()
    try:
        print('connection from', client_address)

        # Receive the data in small chunks and retransmit it
        while True:
            data = connection.recv(16)
            print('received {!r}'.format(data))
            if data:
                print('sending data back to the client')
                connection.sendall(data)
            else:
                print('no data from', client_address)
                break

    finally:
        # Clean up the connection
        connection.close()
```

TCP/IP Client

```
mport socket
import sys

# Tutaj podajemy adres IP własnej maszyny i port
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect the socket to the port where the server is listening
server_address = ('localhost', 10000)
print('connecting to {} port {}'.format(*server_address))
sock.connect(server_address)

try:

    # Send data
    message = b'This is the message.  It will be repeated.'
    print('sending {!r}'.format(message))
    sock.sendall(message)

    # Look for the response
    amount_received = 0
    amount_expected = len(message)

    while amount_received < amount_expected:
        data = sock.recv(16)
        amount_received += len(data)
        print('received {!r}'.format(data))

finally:
    print('closing socket')
    sock.close()
```

Function as thread

```
threading_simple.py
import threading


def worker():
    """thread worker function"""
    print('Worker')


threads = []
for i in range(5):
    t = threading.Thread(target=worker)
    threads.append(t)
    t.start()
```

Threads

```
threading_subclass_args.py
import threading
import logging


class MyThreadWithArgs(threading.Thread):

    def __init__(self, group=None, target=None, name=None,
                 args=(), kwargs=None, *, daemon=None):
        super().__init__(group=group, target=target, name=name,
                         daemon=daemon)
        self.args = args
        self.kwargs = kwargs

    def run(self):
        logging.debug('running with %s and %s',
                      self.args, self.kwargs)


logging.basicConfig(
    level=logging.DEBUG,
    format='(%(threadName)-10s) %(message)s',
)

for i in range(5):
    t = MyThreadWithArgs(args=(i,), kwargs={'a': 'A', 'b': 'B'})
    t.start()
```

9. SMTP

Proste wysyłanie maila zaczerpnięte z https://en.wikibooks.org/wiki/Python_Programming/Email

SMTP

```
mport smtplib
server = smtplib.SMTP('smtp.gmail.com', 587)
server.ehlo()
server.starttls()
server.ehlo()
server.login("youremailusername", "password")
text = msg.as_string()
server.sendmail(fromaddr, toaddr, text)
```

10. Regex

Podpowiedzi:

.	Dowolny znak z wyjątkiem znaku nowej linii	
\d	Dowolna cyfra	
\D	Dowolna znak nie będący cyfrą	
\s	Dowolny biały znak	
\S	Dowolny znak nie będący białym znakiem	
\w	Dowolny znak alfanumeryczny (litera lub cyfra)	Znaki specjalne trzeba poprzedzić '\'
\W	Dowolny znak nie będący znakiem alfanumerycznym	
*	0 lub więcej wystąpień	
+	1 lub więcej wystąpień	
?	0 lub 1 wystąpienie	
r	przed łańcuchem wyłącza specjalne znaczenie backslasha	