 # Primer 1:

```
class Zivotinja:
  # class constructor method
  def __init__(self, ime):

    # instance properties
    self.ime = ime
    self.poslednja_setnja = None
    self.poslednji_obrok = None

    print("Napravljena je nova zivotinja", ime)

  def jedi(self, hrana):
    self.poslednji_obrok = hrana
    print(self.ime, "je pojeo", self.poslednji_obrok)

  def prosetaj(self, mesto = 'nepoznato'):
    print(self.ime, "seta po", mesto)
    self.poslednja_setnja = mesto

  def __str__(self):
        return 'Moje ime je ' + self.ime

# class definition, inherits from Animal
class Pas(Zivotinja):

  # class constructor method
  def __init__(self, ime = "nepoznato", rasa = "nepoznata"):
    # property
    self.rasa = rasa

    # super call, calls the super constructor
    Zivotinja.__init__(self, ime)

# Cat extends Animal
class Macka(Zivotinja):

  def __init__(self, ime, boja):
    self.boja = boja
    Zivotinja.__init__(self, ime)

tom = Macka("Tom", "plavo-bela")
print(tom)

pas = Pas("Rex")
pas.jedi("kost")
print(pas)

# polimorfizam
for i in [pas, tom]:
        i.prosetaj('park')
```

#Primer2:

```
from PySide6.QtWidgets import QApplication, QWidget

# Only needed for access to command line arguments
#import sys

# You need one (and only one) QApplication instance per application.
# Pass in sys.argv to allow command line arguments for your app.
# If you know you won't use command line arguments QApplication([]) works too.
app = QApplication([])#sys.argv)

# Create a Qt widget, which will be our window.
window = QWidget()
window.show()  # IMPORTANT!!!!! Windows are hidden by default.

# Start the event loop.
app.exec()
# Objasniti da event loop zapravo proverava event queue, i ako ima
# novih zahteva prosledjuje ih event heandler koji ih dalje obradjuje

# Your application won't reach here until you exit and the event
# loop has stopped.
```

 # Primer 3:

```
import sys
from PySide6.QtWidgets import QApplication, QPushButton

app = QApplication(sys.argv)

#Posto je ustanovljeno da prozor moze biti bilo sta sto je widget
#https://doc.qt.io/qtforpython/PySide6/QtWidgets/QPushButton.html#qpushbutton
window = QPushButton("Push Me")
window.show()

app.exec()
#Objasniti da ovo nema puno smisla i da se moze koristiti prazan Qwidget u koji se
#ubacuje dodatna funkcionalnosti poput razlicitih tastera i sl.

#Zatim zameniti QPushButton sa QMainWindow (u importu i u instanciranju)
#Navesti da QT obezbedjuje (specijalizovan) prozor sa osnovnim padajucim menijem
#https://doc.qt.io/qtforpython/PySide6/QtWidgets/QMainWindow.html
```

 # Primer 4:

```
import sys
from PySide6.QtWidgets import QApplication,  QMainWindow, QPushButton #,QDockWidget
#Moze se dodati Dock widget
#from PySide6.QtCore import Qt

# Subclass QMainWindow to customize your application's main window
# Dodati ovu klasu na prethodnu pricu
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("My App")
        button = QPushButton("Press Me!")

        #mogu se igrati sa DockWidgetom
        #dockWidget = QDockWidget('Dock')
        #self.addDockWidget(Qt.LeftDockWidgetArea, dockWidget)

        # Set the central widget of the Window.
        self.setCentralWidget(button)
        #self.setMenuWidget(button)

app = QApplication(sys.argv)
window = MainWindow()
window.show()
app.exec()
```

#Primer 5:

```
import sys
from PySide6.QtWidgets import QApplication,  QMainWindow, QPushButton

from PySide6.QtCore import QSize

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("My App")
        button = QPushButton("Press Me!")
# Popricati malo o podesavanju velicine prozora
        self.setFixedSize(QSize(500, 300))
#         self.setMaximumSize(QSize(700, 500))
#         self.setMinimumSize(QSize(200,100))
        self.setCentralWidget(button)

app = QApplication(sys.argv)
window = MainWindow()
window.show()
app.exec()
```