Podstawy Systemów Wbudowanych

Wykład 10: Podstawy programowania w języku Python

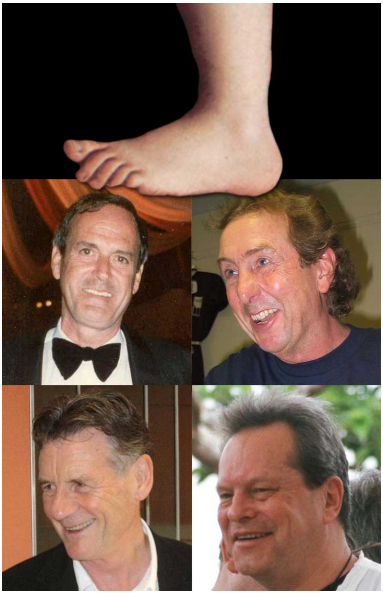
Angelika Tefelska, Dariusz Tefelski

Zakład Fizyki Jądrowej, Wydział Fizyki PW

24 kwietnia 2023

O czym będzie wykład...

1. Dlaczego Python?
2. Typy, zmienne i operacje wejścia-wyjścia
3. Instrukcje warunkowe, pętle
4. Funkcje
5. Klasy i obiekty
6. Operacje na plikach
7. Biblioteki



- Raspberry Pi można oprogramować przy użyciu różnych języków programowania, lecz najbardziej popularnym jest Python. Zresztą drugi człon nazwy Raspberry Pi pochodzi właśnie od Python-a (**P**ython **I**nterpreter.).
- Python został stworzony w 1991 roku przez Guido van Rossum jako następca języka ABC (imperatywny język programowania wykorzystywany do nauczania i tworzenia prototypów).
- Nazwa języka pochodzi od serialu komediowego emitowanego w latach siedemdziesiątych przez BBC: "Latający Cyrk Monty Pythona".
- Głównymi założeniami Pythona było stworzenie języka, który będzie prosty w obsłudze, łatwy w skalowaniu i nie będzie wymuszał zastosowania jednego stylu programowania (programowanie obiektowe, strukturalne, funkcyjne).

Typy zmiennych

Typy proste

- **int** - liczby całkowite:
  >>> **x=24**
- **float** - liczby zmiennoprzecinkowe:
  >>> **x=24.5**
- **complex** - liczby zespolone n. 1+2j:
  >>> **x=complex(1,2)**
- bool - wartości logiczne:
  >>> **a = True**
- str - łańcuchy znaków:
  >>> **napis="Raspberry Pi jest fajne"**
  >>> **napis2='Arduino tez jest fajne'**

Przykład programu w C++

```
#include <iostream>

using namespace std;

int main()
{
        cout<<"Hello World!";
        return 0;
}
```

Przykład programu w Python

```
print("Hello World!")
```

Typy złożone - Lista

Lista

- Lista jest "odpowiednikiem" tablicy ale w przeciwieństwie do tablicy umożliwia przechowywanie różnych typów zmiennych na raz np spis studentów (imię, nazwisko, numer indeksu, czy zarejestrowany na kolejny rok):
  >>>**spis=['Jan', 'Nowak', 224455, True]**

Operacje na listach

- Wyświetlenie tylko pożądanych informacji:
  >>>**print(spis[1])**
  Nowak
  >>> **spis[0:2]**
  ['Jan','Nowak']

Operacje na listach

- Zamiana wartości w liście:
  >>>**spis[0]='Karol'**
  ['Karol','Nowak',224455, True]
- Dodanie elementu na końcu listy:
  >>>**spis.append('Ola')**
  ['Karol','Nowak',224455, True,'Ola']
- Usunięcie elementu z listy: >>> **del spis[4]**
  ['Karol','Nowak',224455, True]
- Dodawanie dwóch list do siebie:
  >>> **osoba=['Ola', 'Kowalska', 232323, True]**
  >>> **spis2=spis+osoba**
  ['Karol','Nowak',224455, True, 'Ola', 'Kowalska', 232323, True]
- Więcej informacji w **dokumentacji**.

Typy złożone - Krotki

Krotka

- Krotka jest to zmienna podobna do listy lecz niemodyfikowalna. Warto ją stosować kiedy nie chcemy aby przypadkowo jakieś wartości zostały zmienione. Przykład deklaracji krotki:

```
>>> x = (1,2,3,4)
```

- Pobieranie konkretnej wartości odbywa się analogicznie jak w przypadku listy czyli:

```
>>> print(x[1])
2
```

Typy złożone - słowniki

Słowniki

Słowniki to taki typ zmiennych, które zawierają klucze oraz przypisane nim wartości np. kluczem może być numer indeksu studenta a wartościami inne dane danego studenta jak imię, nazwisko i zbiór ocen:
lista_studentow={'212121' : ['Jan','Kowalski',[5,4,3,4]],'334455' : ['Anna','Nowak', [3,2,4,5]]}

Operacje na słownikach

- Pobranie elementu ze Słownika:
  >>>**lista_studentow[212121]**
  ['Jan','Kowalski', [5, 4, 3, 4]]
  >>>**lista_studentow[212121][1]**
  Kowalski

Operacje na słownikach

- Nadpisanie lub dodanie nowej wartości do słownika:
  lista_studentow['111111']= ['Ola','Kowalska',[5,5,5,5]]
- Usuwanie elementu ze słownika:
  del lista_studentow['111111']
- Funkcja, która zwraca wszystkie klucze w danym słowniku:
  >>> **lista_studentow.keys()**
  ['212121','334455']
- Funkcja, która usuwa wszystkie dane ze słownika:
  lista_studentow.clear()
- Więcej funkcji w **dokumentacji**

Operacje wejścia-wyjścia

- Wprowadzanie danych do programu przez użytkownika:
  name = input("Podaj swoje imię")
  wiek=int(input("Podaj swój wiek"))
- Wyświetlanie informacji:
  >>> **print("Hello World")**
  Hello World
  >>> **print("Twój wiek to %d" % wiek)**
  Twój wiek to 20
  >>> **print('Witaj 0, twój aktualny wiek to 1'.format(name, wiek)**
  Witaj Karol, twój aktualny wiek to 20

Instrukcje warunkowe

Instrukcje warunkowe

- Instrukcje warunkowe mają składnię:
  if warunek1:
  instrukcje
  elif warunek2:
  instrukcje
  else:
  instrukcje
- Do łączenia warunków służą operatory: and (i), or (lub).

Przykład 1

```
wiek=int(input("Podaj swoj wiek"))

if wiek>18:
        print("Jestes pelnoletni")
```

Przykład 2

```
imie = "Karol"
wiek = 20
if imie == "Karol" and wiek == 20:
        print("Twoje dane:Karol,20lat"
            )
```

Przykład 3

```
wylosowana_liczba=14
if wylosowana_liczba in [20,30]:
        print("Trafiles")
else:
        print("Sprobuj jeszcze raz")
```

Przykład 4

```
import random
numery_lotto=[]
for i in range(0, 6):
        numery_lotto.append(random.
            randint(0,46))
```

Pętle

Pętla for - przykład 1

```
for letter in 'Python':
        print("Current Letter : %s" %
            letter)
```

Pętla for - przykład 2

```
fruits = ['banana', 'apple',  'mango']
for i in range(len(fruits)):
        print("Current fruit : %s" %
            fruits[index])
```

Pętla while - przykład 1

```
n = 100
s = 0
counter = 1

while counter <= n:
        s = s + counter
        counter += 1
```

Pętla for - przykład 3

```
for num in range(0,20):
    for i in range(0,num):
            if num%i == 0:
                    print("Liczba %d jest podzielna przez %d" %(num,i))
            else:
                    print("Liczba %d nie dzieli sie przez %d" %(num,i))
```

Funkcje

Deklaracja

Funkcje deklaruje się w następujący sposób:

```
def nazwa_funkcji(argumety)
        instrukcje_do_wykonania
```

Przykład 1

```
def wynik_mnozenia(x,y)
        return x*y
```

Przykład 2

```
def wynik_mnozenia(x,y=2)
        return x*y
```

Przykład 3

```
def test(*args):
        return args

print test('a', 'abc', 1, 2, 3)
//Otrzymamy: ('a','abc',1,2,3)
```

Przykład 4

```
def test(test, **args):
        return args

print test(test='a', foo='abc', bar
    =1)
//Otrzymamy: {'foo': 'abc', 'bar':
    1}
```

Klasy i obiekty

Deklaracja klasy - przykład 1

```
class MojaKlasa:
        zmienna = "witaj"
        def funkcja(self):
                print "Wiadomosc"
```

Deklaracja obiektu - przykład 2

```
mojobiekt = MojaKlasa()
mojobiekt.zmienna = "hello"
```

Przykład 3

```
class Zespolona:
        def __init__(self,
            rzeczywista, urojona):
                self.r =
                    rzeczywista
                self.i = urojona
        def __str__(self):
                return "( %.1f, %.1
                    f)" %(self.r,
                    self.i)

x = Zespolona(3.0,-4.5)
print(x)
#Otrzymamy: (3.0, -4.5)
```

Dziedziczenie

Klasa nadrzędna

```
class Shape:
        shape_name = None

        def __init __ (self, x=0, y
            =0):
                self.x = x
                self.y = y

        def __str__(self):
                return "%s, x=%g, y
                    =%g" %(self.
                    shape_name,
                    self.x, self.y)
```

Klasa podrzędna

```
class Circle(Shape):
        shape_name = "circle"

        def __init__(self, x=0, y=0
            , radius=1):
                Shape.__init__(self
                    , x, y)
                self.radius =
                    radius

        def __str__(self):
                return Shape.__str
                    __(self) + (",
                    radius=%g" % (
                    self.radius))

        def area(self):
                return self.radius*
                    self.radius
                    *3.14
```

Podstawowe funkcje

- Otworzenie pliku:

  plik=open('ścieżka/plik', tryb)

  Tryby odczytu plików:
  - 'r' - odczyt
  - 'w' - zapis
  - 'a' - dopisanie
  - 'r+' - odczyt i zapis
- Zamknięcie pliku:

  plik.close()
- Odczyt pliku:
  - **plik.read()** - odczyt całego pliku
  - **plik.readline()** - odczyt danej linii wraz ze znakiem końca linii.
  - **plik.readlines()** - zwraca listę wierszy

Operacje na plikach

Przykład 1 - odczyt z pliku

```
plik = open("text.txt", 'r')
for linia in plik:
        print linia
```

Przykład 2 - odczyt z pliku

```
with open('plik.txt') as plik:
        for linia in plik:
                print linia.strip()
                    .split()
//Funkcja split usuwa znaki konca
    linii, spacje itd i zwraca
    liste
```

Przykład 3 - zapis do pliku

```
plik=open("text.txt",'w')
plik.write("Nowa linia danych")
plik.close()
```

Przykład 4 - odczyt danych z CSV

```
import csv
reader = csv.reader(open('/home/cos
    ', 'r'), delimiter=':', quoting
    =csv.QUOTE_NONE)
for row in reader:
        print row
```

Przykład 5 - zapis w formacie CSV

```
import csv
data = [ [1, 'Agata'], [2, 'Karol']
     ]
writer = csv.writer(open('output',
    "w"), dialect=csv.excel)

writer.writerows(data)
```

GPIO Zero

GPIO Zero jest biblioteką zawierającą klasy, które umożliwiają sterowanie zewnętrznymi komponentami takimi jak: dioda, buzzer, silniki itd.

Obsługa diody LED

- Diodę LED można obsłużyć za pomocą klasy: **gpiozero.LED(pin, active_high=True, initial_value=False)**
- Przykład zapalenia diody:

```
from gpiozero import LED

led=LED(17)
led.on()
```

- Metody klasy LED:
  - on() - zapalenie diody
  - off() - zgaszenie diody
  - blink(on_time=1, off_time=1, n=None, background=True)
  - toggle() - zmiana stanu diody np. gdy była zapalona to ją zgasi.
  - is_lit - zwraca informacje czy komponent jest aktywny

GPIO Zero

PWM LED

- Diodą LED można sterować za pomocą PWM korzystając z klasy: **gpiozero.PWMLED(pin, active_high=True, initial_value=0, frequency=100)**
- Przykład zapalenia diody z wypełnieniem 50%:

```
from gpiozero import PWMLED

led=PWMLED(17)
led.value=0.5
led.on()
```

- Metody klasy PWMLED:
  - on() - zapala diodę
  - off() - gasi diodę
  - blink(on_time=1, off_time=1, fade_in_time=0, fade_out_time=0, n=None, background=True)
  - pulse(fade_in_time=1, fade_out_time=1,n=None, background=True) - zmiana jasności świecenie diody o +1/-1
  - toggle() - zmiana stanu diody na przeciwny np. jeśli wypełnienie było 0.4 to zmieni na 0.6

GPIO Zero

RGB LED

- Diodą RGB LED można obsłużyć korzystając z klasy: **gpiozero.RGBLED(red, green, blue, active_high=True, initial_value=(0,0,0), pwm=True)**
- Przykład zaświecenia koloru purpurowego:

```
from gpiozero import RGBLED

led=RGBLED(2,3,4)
led.color(1,0,1)
led.on()
```

- Metody klasy RGBLED:
  - on() - zapalenie diody
  - off() - zgaszenie diody
  - blink(on_time=1, off_time=1, fade_in_time=0, fade_out_time=0, on_color=(1,1,1), off_color=(0,0,0), n=None, background=True)
  - pulse(fade_in_time=1, fade_out_time=1, on_color=(1,1,1), off_color=(0,0,0), n=None, background=True)
  - toggle() - zmiana stanu diody na przeciwny np. gdy początkowy kolor to (0,0,0) to po wykonaniu funkcji będzie (1,1,1)

GPIO Zero

Buzzer

- W celu obsłużenia buzzera należy skorzystać z klasy: **gpiozero.Buzzer(pin, active_high=True, initial_value=False)**
- Przykład włączenie generowania dźwięku:

```
from gpiozero import Buzzer

bz=Buzzer(3)
bz.on()
```

- Metody klasy Buzzer:
  - on() - włączenie Buzzera
  - off() - wyłączenie Buzzera
  - beep(on_time=1, off_time=1, n=None, background=True) - włącza i wyłącza buzzer na przemian
  - toggle() - zmienia stan buzzera np. z włączonego na wyłączony

GPIO Zero

Silnik

- W celu obsłużenia silnika podłączonego przez mostek H należy skorzystać z klasy: **gpiozero.Motor(forward,backward, pwm=True)**
- Przykład obrócenia silnika w położenie przednie:

```
from gpiozero import Motor

motor=Motor(17,18)
motor.forward()
```

- Metody klasy Motor:
  - backward(speed=1)
  - forward(speed=1)
  - stop()

GPIO Zero

Przycisk

- W celu obsłużenia przycisku należy skorzystać z klasy: **gpiozero.Button(pin, pull_up=True, bounce_time=None)**
- Przykład wyświetlenia tekstu po wciśnięciu przycisku:

```
from gpiozero import Button
button = Button(4)
button.wait_for_press()
print("Przycisk zostal wlaczony")
```

- Metody klasy Button:
  - wait_for_press(timeout=None) - pausa do momentu wciśnięcia przycisku
  - wait_for_released(timeout=None) - pausa do momentu wyciśnięcia przycisku
  - when_pressed - funkcja, która wykona się w momencie zmiany stanu z nieaktywnego na aktywny
  - when_released - przeciwna funkcja do poprzedniej
  - when_held - funkcja się wykonuje, gdy przycisk jest przytrzymany przez określoną ilość czasu
  - hold_time - czas przytrzymania
  - is_pressed - zwraca informacje, czy przycisk jest włączony
  - is_held - zwraca informacje czy przycisk jest przytrzymywany

GPIO Zero

Obsługa sensorów

- Biblioteka GPIO Zero udostępnia wiele klas do obsługi sensorów. Poniżej przykłady kilku z nich
- Przykład: wyświetlenie komunikatu w zależności czy sensor wykrył linię:

```
from gpiozero import LineSensor
from signal import pause

sensor = LineSensor(4)
sensor.when\_line = lambda : print("Linia wykryta")
sensor.when\_no\_line = lambda : print("Brak linii")
pause()
```

- Przykład: obsługa czujnika ruchu:

```
from gpiozero import MotionSensor

pir=MotionSensor(4)
pir.wait\_for\_motion()
print("Ruch wykryty")
```

RPI.GPIO

Podstawowe funkcje

- RPI.GPIO służy do obsługi samego GPIO a nie jak w przypadku GPIO Zero różnych komponentów.
- W celu skorzystania z biblioteki RPI.GPIO należy ją zaimportować: **import RPI.GPIO as GPIO**
- Numeracja pinów może być: BOARD (nazwy z płytki) lub BCM (numeracja z samego procesora). Aby wybrać dany typ należy skorzystać z funkcji: **GPIO.setmode(GPIO.BOARD lub GPIO.BCM)**
- W celu określenia czy dany pin ma być wejściowy czy wyjściowy należy skorzystać z funkcji: **GPIO.setup(channel, GPIO.IN lub GPIO.OUT)**
- Biblioteka również umożliwia ustawienie za jednym razem kilka pinów jako wejściowe lub wyjściowe: **GPIO.setup(lista z numerami pinów, GPIO.OUT lub GPIO.IN)**
- Odczytanie danych z wejścia: **GPIO.input(channel)**
- W celu ustawienia konkretnej wartości na wyjściu należy skorzystać z funkcji: **GPIO.output(channel, state np. GPIO.LOW)**
- Wyczyszczenie ustawień ze wszystkich pinów: **GPIO.cleanup()** lub z wybranych: **GPIO.cleanup(channel)** albo **GPIO.cleanup([channel1, channel2])**.

RPI.GPIO

Piny wejściowe

- Biblioteka zawiera kilka przydatnych funkcji przy obsłudze pinów wejściowych takich jak: **wait_for_edge()** oraz **event_detected()**.
- Funkcja do wykrywania zbocza ma następującą postać: **GPIO.wait_for_edge(channel, GPIO.RISING lub GPIO.FALLING lub GPIO.BOTH)**
- Przykład: poczekaj 5s i wyświetl komunikat czy zbocze zostało wykryte:

```
import RPI.GPIO as GPIO
channel = GPIO.wait_for_edge(channel, GPIO_RISING, timeout=5000)
if channel is None:
        print('Timeout occurred')
else:
        print('Edge detected on channel', channel)
```

RPI.GPIO

RPI.GPIO

- Funkcja **event_detected()** działa analogicznie jak **wait_for_edge()** ale w przeciwieństwie do niej chodzi w tle niezależnie od głównej pętli. Pozwala to uniknąć przeoczenia sygnału szczególnie gdy wykorzystujemy biblioteki typu PyQt, w których w pętli jest ciągłe odświeżanie GUI.
- Przykład:

```
GPIO.add_event_detect(channel, GPIO.RISING)
do_something()
if GPIO.event_detected(channel):
        print('Button pressed')
```

RPI.GPIO

Obsługa PWM-a

- W celu skorzystania z PWM należy skonfigurować dany pin korzystając z: **p = GPIO.PWM(channel, frequency)**
- Następnie należy ustawić wypełnienie: **p.start(wypełnienie od 0 do 100)**
- Częstotliwość można zmienić za pomocą funkcji: p.ChangeFrequency(częstotliwość w Hz)
- Wypełnienie można zmienić za pomocą funkcji: **p.ChangeDutyCycle(wypełnienie)**
- Zatrzymanie generowanie sygnału PWM można zrealizować za pomocą funkcji: **p.stop()**

WiringPi

Biblioteka WiringPi

- Oprócz bibliotek opartych na Python, istnieją również biblioteki do C/C++. Taką biblioteką jest właśnie WiringPi, który do złudzenia przypomina biblioteki do arduino.
- Przykład: mrugająca dioda co 500ms.

```
#include <wiringPi.h>
int main (void)
{
        wiringPiSetup();
        pinMode(0, OUTPUT) ;
        for (;;)
        {
                digitalWrite(0, HIGH);
                delay(500);
                digitalWrite(0,  LOW);
                delay(500);
        }
        return 0 ;
}
```

- Więcej na temat biblioteki na stronie

WiringPi

Obsługa w bash-u

- Korzystając z WiringPi można obsłużyć GPIO korzystając z bash-a. Podstawowe komendy to:
  - gpio readall - zwraca informacje o wszystkich pinach
  - gpio -g mode numer_pinu out/in - ustawia dany pin jako wejściowy lub wyjściowy
  - gpio -g write numer_pinu wartość - ustawia wartość na danym pinie z zakresu (0,1)
  - gpio -g read numer_pinu - umożliwia odczyt wartości z danego pinu
  - gpio -g mode numer_pinu pwm - ustawienie trybu pwm
  - gpio -g pwm numer_pinu wypenienie - ustawienie wypełnienia z zakresu (0,1023)
  - gpio -g mode numer_pinu down/up - ustawienie rezystorów podciągających

WiringPi

Przykład: zapalenie diody (pin 17) po naciśnięciu przycisku (pin 23).

```
#!/bin/bash

gpio -g mode 23 in
gpio -g mode 17 out
gpio -g mode 23 down

while true; do
x='gpio -g read 23'
if [ $x -ge 1 ]
then
gpio -g write 17 1
else
gpio -g write 17 0
sleep 0.2
fi

done
```

PIGPIO

Obsługa w bash-u

- Bibliotekę PIGPIO można obsłużyć zarówno pod C/C++ jak i Python-em przy włączonym pigpio daemon. Dokumentacja wraz z przykładami jest dostępna na stronie.
- Z biblioteki pigpio można też skorzystać pod bash-em. Podstawowe komendy to:
  - pigs m numer_pinu o/i - ustawienie danego pinu jako wyjściowego (o) lub wejściowego (i)
  - pigs w numer_pinu wartość - ustawienie danej wartości na pinie z zakresu (0,1)
  - pigs r numer_pinu - odczyt wartości z pinu
  - pigs pud numer_pinu u/d - podciągnięcie rezystorów
  - pigs s numer_pinu wypełnienie - funkcja do sterowanie serwomechanizmem, w której wypełnienie przyjmuje wartości z zakresu (500,2500)
  - pigs p numer_pinu wypełnienie - generowanie sygnału PWM o wypełnieniu z zakresu (0,255)

THAT'S ALL FOR TODAY....
ANY QUESTION??