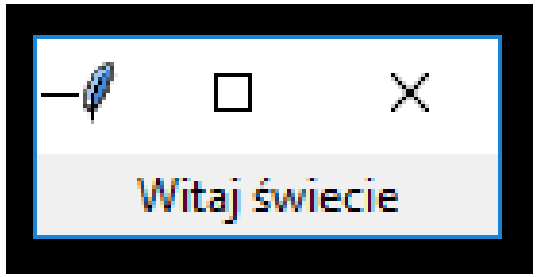
Python
wykład 5

Graficzny interfejs użytkownika dla Pythona

Python jako taki nie został zaprojektowany z myślą o GUI – jest z założenia językiem skryptowym uruchamianym w konsoli. Oczywiście jak już się przekonaliśmy, to że czegoś nie ma „fabrycznie” nie oznacza, że nie można tego dodać później. I oczywiście dodano – w tzw. bibliotece standardowej pojawił się moduł tkinter (pierwotnie tk_inter), który pozwala tworzyć proste aplikacje, wykorzystując podstawowe elementy GUI. Oczywiście są zewnętrzne biblioteki pozwalające na znacznie więcej niż tkinter (np. PyQt, wxPython, kivy itp.), ale standardową bibliotekę warto znać.

TkInter 01 – witaj świecie

```
1  from tkinter import *
2  root = Tk()
3  myLabel = Label(root,text="Witaj świecie")
4  myLabel.pack()
5  root.mainloop()
```



TkInter de facto jest modułem, który udostępnia zestaw funkcji, stałych i obiektów, na podstawie, których tworzymy widgety odpowiadające za poszczególne kontrolki.
Generalnie program GUI używający TkInter składa się zazwyczaj z
następujących czterech elementów:

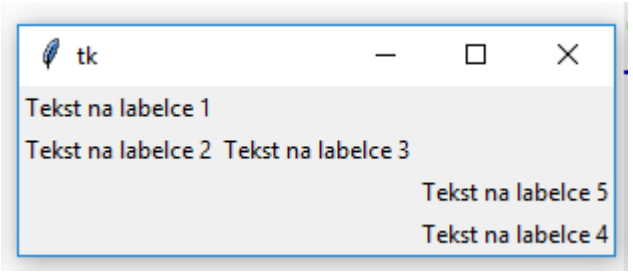
1. importu modułu tkinter
2. stworzenia okna głównego programu
3. dodania do okna koniecznych „widżetów”
4. uruchomienia kontrolera zdarzeń

1 - najprostsza metoda - importujemy wszystko z pakietu tkinter - nie musimy ale możemy
2 - pierwszą rzeczą od której zaczynamy jest przygotowanie głównego (root) widgetu (generalnie jakiegoś okna). Jest to obiekt klasy Tk.
3 – klasyczny element typu label umożliwiający wyświetlenie jakiegoś tekstu. Zawartość interfejsu generujemy w sposób dynamiczny zatem najpierw zawsze tworzymy widget o potem "umieszczamy" go na ekranie. W tym wypadku tworzymy obiekt myLabel, który jest obiektem klasy Label, jako parametry konstruktora przekazujemy widget rodzica (w tym wypadku root) oraz tekst, który ma zostać wyświetlony.

4 – umieszczamy przygotowany wcześniej widget na oknie. Metoda pack() jest najprostszym organizatorem układu – domyślnie umieszcza kontrolkę pod wcześniej dodanymi.
5 - pętla główna kontrolera zdarzeń – to proces, który odpowiedzialny jest z wyświetlanie interfejsu, jego aktualizowanie i reagowanie na zdarzenia

TkInter 02 – układ grid

```
1   from tkinter import *
2   root = Tk()
3
4   myLabel1 = Label(root,text="Tekst na labelce 1")
5   myLabel2 = Label(root,text="Tekst na labelce 2")
6   myLabel3 = Label(root,text="Tekst na labelce 3")
7   myLabel4 = Label(root,text="Tekst na labelce 4")
8   myLabel1.grid(row=0, column=0)
9   myLabel2.grid(row=1, column=0)
10  myLabel3.grid(row=1, column=1)
11  myLabel4.grid(row=5, column=5)
12  myLabel5 = Label(root,text="Tekst na labelce 5").grid(row=4, column=5)
13
14  root.mainloop()
```

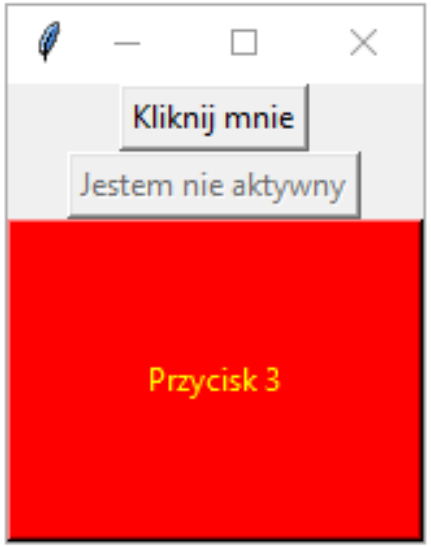


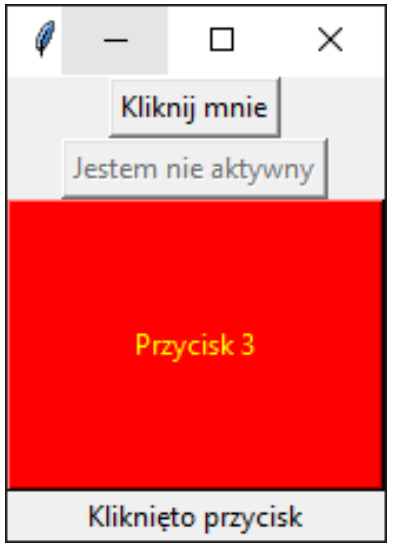
W przykładzie poprzednim wykorzystaliśmy menagera układu u nazwie pack(). Dodawał on nowy element interfejsu pod poprzednimi. Tym razem skorzystamy z menagera grid(), który umożliwia rozkładanie elementów interfejsu w oparciu o swego rodzaju siatkę (tabelę), której elementy są określane poprzez wskazanie numeru wiersza i kolumny.
4 – 7 – tworzymy 4 kolejne labelki dokładnie tak samo jak w przypadku poprzednim.

8 – układamy pierwszą labelkę – wiersze i kolumny numerowane są od 0, zatem zostanie ona wyświetlona w lewym górnym rogu obszaru.
9-11 – rozmieszczamy kolejne labelki na obszarze okna, przy czym należy pamiętać, że grid działa trochę „relatywnie” - jeśli ustawimy coś w kolumnie 5 ale po drodze nie będzie kolumn 2,3 i 4 to nie znaczy, ze pojawią się jakieś puste przestrzenie - w przykładzie labelka zostanie umieszczona obok wcześniejszych elementów
12 – ponieważ jak pamiętamy w python wszystko jest obiektem, zatem powstały w wyniku działania obiekt od razu możemy „poprosić” o wykonanie jakiejś metody – np. ułożenie w grid

TkInter 03 – przyciski i marginesy

```
1   from tkinter import *
2
3   root= Tk()
4
5   def obslugaKlikniecia():
6       nowaLabelka = Label(root, text="Kliknięto przycisk")
7       nowaLabelka.pack()
8
9   myButton = Button(root, text="Kliknij mnie", command=obslugaKlikniecia)
10  myButton2 = Button(root, text="Jestem nie aktywny", state=DISABLED)
11  myButton3 = Button(root, text="Przycisk 3", padx=50, pady=50, fg="yellow"
12
13  myButton.pack()
14  myButton2.pack()
15  myButton3.pack()
16
17  root.mainloop()
```





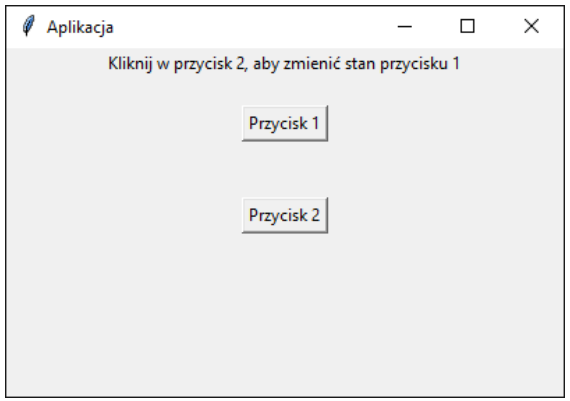
5 – 7 – prosta funkcja, która będzie reagować na naciśnięcie przycisku (ją podepniemy jako tzw. procedurę obsługi. Funkcja tworzy nową labelkę (6) i umieszcza ją pod wcześniej istniejącymi elementami

9 – nowy element klasy Button czyli klasyczny przycisk. Oczywiście jak każdy inny element interfejsu musi mieć określonego rodzica (root). Przydałby mu się również jakiś wyświetlany tekst („Kliknij mnie”) no i oczywiście podpięta funkcja, która będzie wykonywana, gdy przycisk zostanie kliknięty (command). Warto zwrócić uwagę na to, że my tutaj ustalamy wartość atrybutu command – tak jakbyśmy przypisywali wskaźnik do funkcji – nie ma w tym przypadku obslugaKlikniecia(), bo nawiasy na końcu oznaczałyby automatyczne wywołanie metody, a my tego nie chcemy – ma się ona wywołać dopiero jak nastąpi kliknięcie

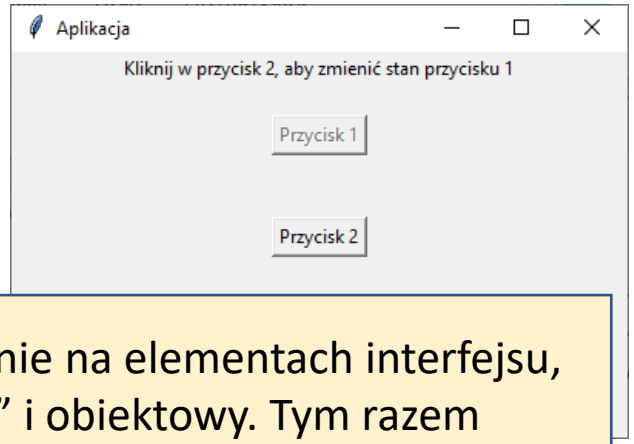
10 – przycisk jak wyżej – nie jest klikalny nie podłączamy mu obsługi kliknięcia,

11 – ponownie przycisk, dodatkowo ustawiamy marginesy, kolor tła oraz kolor czcionki

TkInter 04 – graficznie i obiektowo, cz. 1




```
1  from tkinter import *
2  class TAplikacja:
3      def __init__(self, rodzic):
4          self.rodzic = rodzic
5          self.lKomentarz = Label(self.rodzic, text="Kliknij w przycisk 2, aby zmienić stan przycisku 1")
6          self.lKomentarz.pack()
7          self.przycisk1 = Button(self.rodzic, text = "Przycisk 1", state = NORMAL)
8          self.przycisk1.pack(pady = 20)
9          self.przycisk2 = Button(self.rodzic, text = "Przycisk 2", command = self.zmienStan)
10         self.przycisk2.pack(pady = 20)
11     def zmienStan(self):
12         if (self.przycisk1['state'] == NORMAL):
13             self.przycisk1['state'] = DISABLED
14         else:
15             self.przycisk1['state'] = NORMAL
16
17 root = Tk()
18 root.title("Aplikacja")
19 root.geometry("400x250")
20 aplikacja = TAplikacja(root)
21 root.mainloop()
```



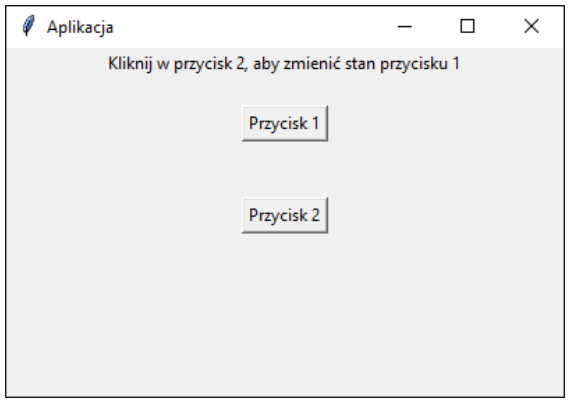
W naszych przykładach skupiamy się obecnie na elementach interfejsu, a nie na tym, żeby nasz kod był „elegancki” i obiektowy. Tym razem zróbmy coś jednak „porządnie”

2 – początek klasy, która będzie odpowiadać zawartości naszego okna

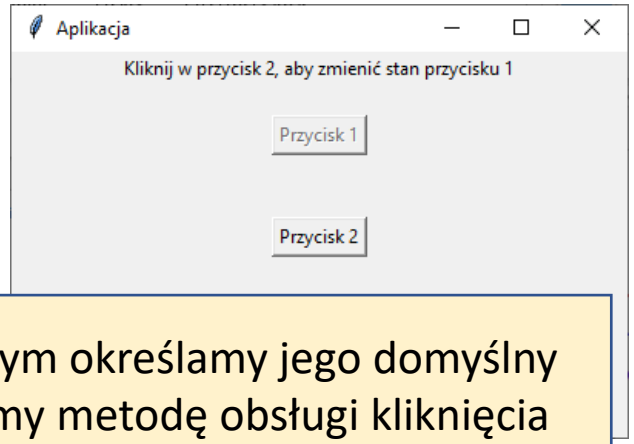
3, 4 – początek konstruktora naszej klasy – jako parametr oczywiście otrzymuje self (to już było wcześniej) oraz okno, które będzie rodzicem (na nim będziemy wyświetlać tworzone elementy)

5, 6 – labelka z komentarzem. Warto zwrócić uwagę na pierwszy argument konstruktora – labelka będzie umieszczona na self.rodzic, czymkolwiek on będzie

TkInter 04 – graficznie i obiektowo, cz. 2

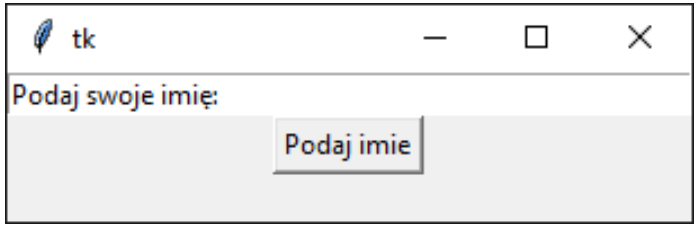


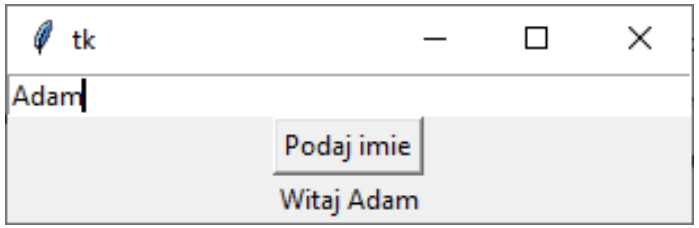

```
1  from tkinter import *
2  class TAplikacja:
3      def __init__(self, rodzic):
4          self.rodzic = rodzic
5          self.lKomentarz = Label(self.rodzic, text="Kliknij w przycisk 2, aby zmienić stan przycisku 1")
6          self.lKomentarz.pack()
7          self.przycisk1 = Button(self.rodzic, text = "Przycisk 1", state = NORMAL)
8          self.przycisk1.pack(pady = 20)
9          self.przycisk2 = Button(self.rodzic, text = "Przycisk 2", command = self.zmienStan)
10         self.przycisk2.pack(pady = 20)
11     def zmienStan(self):
12         if (self.przycisk1['state'] == NORMAL):
13             self.przycisk1['state'] = DISABLED
14         else:
15             self.przycisk1['state'] = NORMAL
16
17 root = Tk()
18 root.title("Aplikacja")
19 root.geometry("400x250")
20 aplikacja = TAplikacja(root)
21 root.mainloop()
```



7 – 10 – dodajmy dwa przyciski, w pierwszym określamy jego domyślny stan (***state=NORMAL***), w drugim podpinamy metodę obsługi kliknięcia (***self.zmienStan*** – jest to w końcu metoda naszej klasy)
11 – 15 – metoda klasy ***TAplikacja*** – sprawdza aktualny stan przycisku 1 i w zależności od tego zmienia jest stan przeciwny (***NORMAL*** lub ***DISABLED***)
17 – 19 - tworzymy okno główne (w przykładzie ***root***) . Ustalamy wyświetlany w nim tytuł oraz określamy jego rozmiary
20 – tworzymy nowy obiekt klasy ***TAplikacja***, przekazujemy do jej konstruktora wcześniej przygotowane okno główne (***root***)

TkInter 05 – pole tekstowe



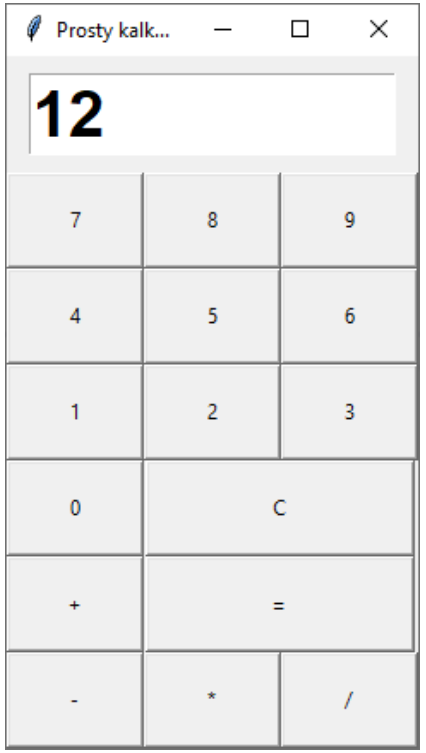



```
1   from tkinter import *
2   root = Tk()
3
4   def obslugaKlikniecia():
5       nowaLabelka.config(text="Witaj "+poleTekstowe.get())
6
7   poleTekstowe = Entry(root, width=50)
8   poleTekstowe.pack()
9   poleTekstowe.insert(0,"Podaj swoje imię: ")
10
11  przycisk = Button(root, text="Podaj imie", command=obslugaKlikniecia)
12  przycisk.pack()
13
14  nowaLabelka = Label(root, text="")
15  nowaLabelka.pack()
16  root.mainloop()
```

4, 5 – prosta metoda, która wyświetlać będzie tekst wpisany przez użytkownika w polu tekstowym. Zawartość labelki modyfikujemy przy użyciu metody ***config()*** gdzie jako argument przekazujemy nazwę atrybutu (***text***) oraz jego wartość – w tym wypadku będzie to efekt działania metody ***get()*** dla obiektu ***poleTekstowe***
7, 8 – tworzymy i wyświetlamy pole tekstowe – jest to obiekt klasy ***Entry***, jego szerokość została ustalona na 50 (nie 50 px tylko 50 znaków)
9 – obiekty klasy ***Entry*** posiadają metodą ***insert()***, która pozwala wstawić tekst w nich wyświetlany
11 – 15 – tworzymy i wyświetlamy przycisk oraz labelkę

TkInter 06 – przykład praktyczny cz. 1.1

```
1   from tkinter import *
2   oknoGlowne = Tk()
3
4   oknoGlowne.title("Prosty kalkulator")
5
6   poleTekstowe = Entry(oknoGlowne, width=10, font= ('Arial 28 bold'))
7   poleTekstowe.grid(row=0, column=0, columnspan=3, padx=10, pady=10)
8
9   def obslugaPrzycisku(symbol):
10      aktWartosc= str(poleTekstowe.get())
11      poleTekstowe.delete(0,END)
12      poleTekstowe.insert(0,aktWartosc+str(symbol))
13
14  def obslugaPrzyciskuCzysc():
15      poleTekstowe.delete(0,END)
16
17  def obslugaPrzyciskuDodaj():
18      global globPopWartosc
19      globPopWartosc=int(poleTekstowe.get())
20      global operacja
21      operacja="dodawanie"
22      poleTekstowe.delete(0,END)
23
24  def obslugaPrzyciskuOdejmij():
25      global globPopWartosc
26      globPopWartosc=int(poleTekstowe.get())
27      global operacja
28      operacja="odejmowanie"
29      poleTekstowe.delete(0,END)
30
```



Czas na ulubione zadanie każdego studenta – kalkulator

6 – pole tekstowe, w którym będą wyświetlane wprowadzone cyfry oraz wyniki działania – w tym wypadku zmieniamy domyślną czcionkę, aby wyświetlacz był lepiej widoczny

7 – kalkulator jako całość „rozpięty” będzie w oparciu o układ typu grid z 3 kolumnami, pole tekstowe zostało rozciągnięta na 3 columnach (***columnspan=3***)

9 – metoda, która będzie wykorzystana do obsługi kliknięcia wszystkich przycisków z cyframi (jako parametr przesyłamy odpowiedni znak)

TkInter 06 – przykład praktyczny cz. 1.2

```
1   from tkinter import *
2   oknoGlowne = Tk()
3
4   oknoGlowne.title("Prosty kalkulator")
5
6   poleTekstowe = Entry(oknoGlowne, width=10, font= ('Arial 28 bold'))
7   poleTekstowe.grid(row=0, column=0, columnspan=3, padx=10, pady=10)
8
9   def obslugaPrzycisku(symbol):
10      aktWartosc= str(poleTekstowe.get())
11      poleTekstowe.delete(0,END)
12      poleTekstowe.insert(0,aktWartosc+str(symbol))
13
14  def obslugaPrzyciskuCzysc():
15      poleTekstowe.delete(0,END)
16
17  def obslugaPrzyciskuDodaj():
18      global globPopWartosc
19      globPopWartosc=int(poleTekstowe.get())
20      global operacja
21      operacja="dodawanie"
22      poleTekstowe.delete(0,END)
23
24  def obslugaPrzyciskuOdejmij():
25      global globPopWartosc
26      globPopWartosc=int(poleTekstowe.get())
27      global operacja
28      operacja="odejmowanie"
29      poleTekstowe.delete(0,END)
30
```

10 – 12 – zapamiętujemy w zmiennej pomocniczej aktualnie wyświetlany ciąg (***poleTekstowe.get()***), czyścimy jego zawartość (***wiersz 11***), a następnie ustalamy jego zwartość jako połączenie poprzedniej zawartości oraz aktualnego znaku

14, 15 – metoda czyszcząca zawartość „wyświetlacza”

17 – obsługa przycisku dodawania

18, 19 – nowa zmienna o zasięgu globalnym – w niej zapamiętamy co było na wyświetlaczu w momencie naciskania na przycisk równości (dla ułatwienia ograniczamy się do liczb całkowitych)

20, 21 – kolejna zmienna globalna - będzie „pamiętać” jaka aktualnie operacja jest wykonywana – w tym wypadku oczywiście przyjmie wartość „dodawanie”

22 – w każdym kalkulatorze po naciśnięciu symbolu operacji wyświetlacz jest kasowany – my również tak zrobimy

24 – 29 – metoda analogiczna jak w przypadku dodawania – różni się tylko tym, jaką wartość wpisuje do globalnej zmiennej operacja

TkInter 06 – przykład praktyczny cz. 2

```
31  def obslugaPrzyciskuPomnoz():
32      global globPopWartosc
33      globPopWartosc=int(poleTekstowe.get())
34      global operacja
35      operacja="mnozenie"
36      poleTekstowe.delete(0,END)
37
38  def obslugaPrzyciskuPodziel():
39      global globPopWartosc
40      globPopWartosc=int(poleTekstowe.get())
41      global operacja
42      operacja="dzielenie"
43      poleTekstowe.delete(0,END)
44
45  def obslugaPrzyciskuRowny():
46      aktualnaWartosc = poleTekstowe.get()
47      poleTekstowe.delete(0,END)
48      if operacja=="dodawanie":
49          poleTekstowe.insert(0,globPopWartosc+int(aktualnaWartosc))
50      if operacja=="odejmowanie":
51          poleTekstowe.insert(0,globPopWartosc-int(aktualnaWartosc))
52      if operacja=="mnozenie":
53          poleTekstowe.insert(0,globPopWartosc*int(aktualnaWartosc))
54      if operacja=="dzielenie":
55          poleTekstowe.insert(0,globPopWartosc/int(aktualnaWartosc))
```

31 – 43 – kolejne dwie metody obsługi operacji – działają identycznie jak poprzednie tylko dotyczą mnożenia i dzielenia
45 – metoda obsługująca przycisk równości
46, 47 – podobnie jak wcześniej zapamiętujemy w zmiennej pomocniczej aktualną zawartość wyświetlacza, a następnie czyścimy go
48 – 55 – w zależności od tego, jaka było ostatnio wybrana operacja na wyświetlaczu powinien pojawić się wynik dodawanie, odejmowanie, mnożenia lub dzielenia. Warto zauważyć, że metoda ***insert()*** „radzi sobie” z różnymi typami danych (nie musimy ich konwertować na napis, co nie znaczy, że nie powinniśmy tak dla porządku ☺)

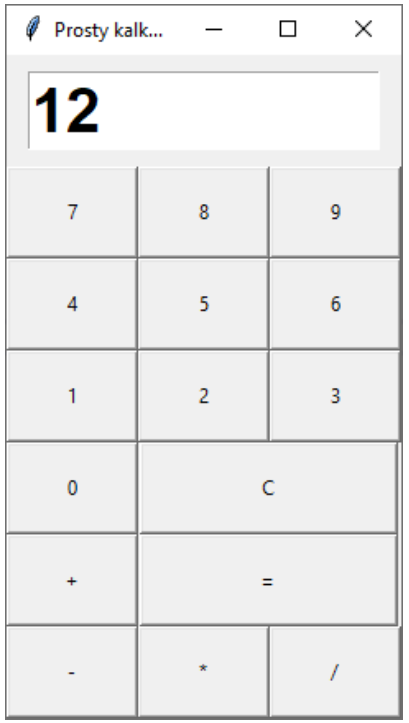
TkInter 06 – przykład praktyczny cz. 3

```
56
57  przycisk_1 = Button(oknoGlowne, text="1", width=10, height=3, command=lambda: obslugaPrzycisku(1))
58  przycisk_2 = Button(oknoGlowne, text="2", width=10, height=3, command=lambda: obslugaPrzycisku(2))
59  przycisk_3 = Button(oknoGlowne, text="3", width=10, height=3, command=lambda: obslugaPrzycisku(3))
60  przycisk_4 = Button(oknoGlowne, text="4", width=10, height=3, command=lambda: obslugaPrzycisku(4))
61  przycisk_5 = Button(oknoGlowne, text="5", width=10, height=3, command=lambda: obslugaPrzycisku(5))
62  przycisk_6 = Button(oknoGlowne, text="6", width=10, height=3, command=lambda: obslugaPrzycisku(6))
63  przycisk_7 = Button(oknoGlowne, text="7", width=10, height=3, command=lambda: obslugaPrzycisku(7))
64  przycisk_8 = Button(oknoGlowne, text="8", width=10, height=3, command=lambda: obslugaPrzycisku(8))
65  przycisk_9 = Button(oknoGlowne, text="9", width=10, height=3, command=lambda: obslugaPrzycisku(9))
66  przycisk_0 = Button(oknoGlowne, text="0", width=10, height=3, command=lambda: obslugaPrzycisku(0))
67
68  przycisk_dod = Button(oknoGlowne, text="+", width=10, height=3, command=obslugaPrzyciskuDodaj)
69  przycisk_odej = Button(oknoGlowne, text="-",width=10, height=3, command=obslugaPrzyciskuOdejmij)
70  przycisk_mnoz = Button(oknoGlowne, text="*", width=10, height=3, command=obslugaPrzyciskuPomnoz)
71  przycisk_dziel = Button(oknoGlowne, text="/", width=10, height=3, command=obslugaPrzyciskuPodziel)
72
73  przycisk_row = Button(oknoGlowne, text="=", width=21, height=3, command=obslugaPrzyciskuRowny)
74  przycisk_czysc = Button(oknoGlowne, text="C", width=21, height=3, command=obslugaPrzyciskuCzysc)
```

57 – 74 – dość prosty kod, w którym kolejno tworzymy kolejne przyciski kalkulatora. Warto zwrócić uwagę, że w przypadku przycisków z cyframi, wykorzystujemy tę samą metodę obsługi kliknięcia, musimy zatem w jakiś sposób przekazać jej parametr sterujący. Nie możemy jednak po prostu napisać nazwy metody z parametrem przekazanym w nawiasach. Spowoduje to automatyczne uruchomienie metody, a ma się ona uruchamiać dopiero w odpowiednim momencie. Wykorzystaliśmy tutaj jednak wyrażenie ***lambda***, tworząc de facto nową metodą anonimową i de facto ją przypisujemy jako wartość ***command*** dla każdego z przycisków

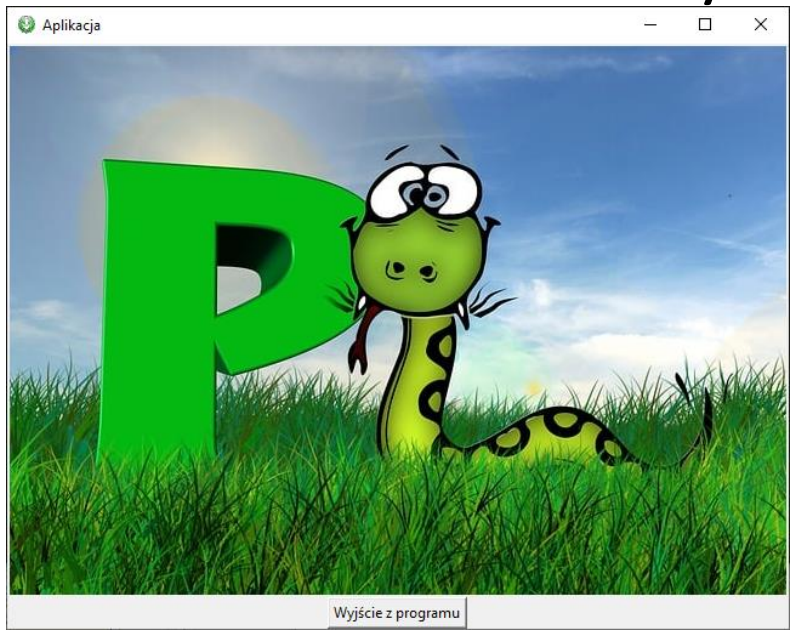
TkInter 06 – przykład praktyczny cz. 4

```
76      przycisk_1.grid(row=3 , column=0 )
77      przycisk_2.grid(row=3 , column=1 )
78      przycisk_3.grid(row=3 , column=2 )
79      przycisk_4.grid(row=2 , column=0 )
80      przycisk_5.grid(row=2 , column=1 )
81      przycisk_6.grid(row=2 , column=2 )
82      przycisk_7.grid(row=1 , column=0 )
83      przycisk_8.grid(row=1 , column=1 )
84      przycisk_9.grid(row=1 , column=2 )
85      przycisk_0.grid(row=4 , column=0 )
86
87      przycisk_dod.grid(row=5, column=0)
88      przycisk_czysc.grid(row=4, column=1, columnspan=2)
89      przycisk_row.grid(row=5, column=1, columnspan=2)
90
91      przycisk_odej.grid(row=6, column=0)
92      przycisk_mnoz.grid(row=6, column=1)
93      przycisk_dziel.grid(row=6, column=2)
94
95      oknoGlowne.mainloop()
```



76 – 93 – korzystając z grid rozmieszczamy wszystkie przyciski kalkulatora

TkInter 07 – ikony i grafika



2 – importujemy moduły, które pozwolą nam na obsługę plików graficznych
6 – zmieniamy domyślną ikoną okna tkintera – w przykładzie zakładamy, że down.ico znajduje się w tym samym folderze co skrypt
7 – tworzymy nowy obiekty klasy ***PhotoImage***. Jako parametr konstruktora przekazujemy nazwę pliku graficznego, który ma zostać wyświetlony
8 – dodajemy do okienka obiekt klasy ***Label*** – tym razem jako parametr konstruktora przekazujemy wartość atrybutu image. Jest nim ***mojObraz***, który został utworzony w wierszu **7**
10 – dodatkowy przycisk, który pozwoli zamknąć okno główne (i cały program) – korzystamy z metody **quit** wywoływanej na rzecz okna, które chcemy zamknąć

```
1   from tkinter import *
2   from PIL import ImageTk, Image
3   oknoGlowne = Tk()
4
5   oknoGlowne.title('Aplikacja')
6   oknoGlowne.iconbitmap('down.ico')
7   mojObraz = ImageTk.PhotoImage(Image.open("python.jpg"))
8   labelWyswObraz = Label(image=mojObraz)
9   labelWyswObraz.pack()
10  przyciskKoniec = Button(oknoGlowne, text="Wyjście z programu", command = oknoGlowne.quit)
11  przyciskKoniec.pack()
12  oknoGlowne.mainloop()
```

TkInter 08 – przeglądarka zdjęć cz. 1.1

3 – klasa podstawowa naszej aplikacji (tym razem tak trochę „ładniej”)
4 – nagłówek konstruktora – przekazujemy dwa parametry widget na którym aplikacja będzie wyświetlana (***oknoRodzica***) oraz listę nazw plików, które mają być wyświetlane.
5 – ustalamy wartość atrybutu okna rodzica
6 – atrybut pomocniczy, w którym przechowywać będziemy numer aktualnie wyświetlanego zdjęcia
7 – lista nazw obrazów, które mają być wyświetlane

```
1   from tkinter import *
2   from PIL import ImageTk, Image
3   class TPrzegladarka:
4       def __init__(self, oknoRodzica,nazwyObrazow):
5           self.oknoRodzica = oknoRodzica
6           self.aktFoto=0
7           self.listaNazwObrazow=nazwyObrazow
8           self.listaObrazow=[]
9           for nazwa in self.listaNazwObrazow:
10              self.listaObrazow.append(ImageTk.PhotoImage(Image.open("zdjecia/"+nazwa)))
11
12          self.labelStatus = Label(oknoRodzica, text='testowa', bd=1,  relief=SUNKEN, anchor=E)
13          self.labelStatus.grid(row=2, column=0, columnspan=3, sticky=W+E)
14
15          self.labelWyswObraz = Label(image=self.listaObrazow[0])
16          self.labelWyswObraz.grid(row=0, column=0, columnspan=3)
17
18          self.przyciskWstecz = Button(self.oknoRodzica, text="Poprzedni", command=self.wyswietlPoprzedni, state=NORMAL)
19          self.przyciskWstecz.grid(row=1, column=0)
20          self.przyciskKoniec = Button(self.oknoRodzica, text="Wyjście z programu",  command = self.oknoRodzica.quit)
21          self.przyciskKoniec.grid(row=1, column=1)
22          self.przyciskNastepny = Button(self.oknoRodzica, text="Nastepny", command=self.wyswietlNastepny)
23          self.przyciskNastepny.grid(row=1, column=2)
24          self.wyswietlAktualny()
```

TkInter 08 – przeglądarka zdjęć cz. 1.2

8 – lista obiektów klasy ***PhotoImage*** – w niej będziemy przechowywali gotowe obrazy załadowane wcześniej z plików. Oczywiście zwiększa to ilość pamięci zużytej w czasie działania skryptu, z drugiej – przyspiesza jego działanie (nie odczytujemy plików w czasie przeglądania).
9, 10 – pętla, która przegląda listę z nazwami obrazów i przy założeniu, że znajdują się w podfolderze zdjęcia (względem folderu, w którym jest skrypt), tworzy nowe obiekty klasy ***TimageTk.PhotoImage*** na ich podstawie. Każdy z tak utworzonych obrazów, dopisywany jest następnie do wcześniej przygotowanej listy (***self.listaObrazow.append()***)

```
1   from tkinter import *
2   from PIL import ImageTk, Image
3   class TPrzegladarka:
4       def __init__(self, oknoRodzica,nazwyObrazow):
5           self.oknoRodzica = oknoRodzica
6           self.aktFoto=0
7           self.listaNazwObrazow=nazwyObrazow
8           self.listaObrazow=[]
9           for nazwa in self.listaNazwObrazow:
10              self.listaObrazow.append(ImageTk.PhotoImage(Image.open("zdjecia/"+nazwa)))
11
12          self.labelStatus = Label(oknoRodzica, text='testowa', bd=1,  relief=SUNKEN, anchor=E)
13          self.labelStatus.grid(row=2, column=0, columnspan=3, sticky=W+E)
14
15          self.labelWyswObraz = Label(image=self.listaObrazow[0])
16          self.labelWyswObraz.grid(row=0, column=0, columnspan=3)
17
18          self.przyciskWstecz = Button(self.oknoRodzica, text="Poprzedni", command=self.wyswietlPoprzedni, state=NORMAL)
19          self.przyciskWstecz.grid(row=1, column=0)
20          self.przyciskKoniec = Button(self.oknoRodzica, text="Wyjście z programu",  command = self.oknoRodzica.quit)
21          self.przyciskKoniec.grid(row=1, column=1)
22          self.przyciskNastepny = Button(self.oknoRodzica, text="Nastepny", command=self.wyswietlNastepny)
23          self.przyciskNastepny.grid(row=1, column=2)
24          self.wyswietlAktualny()
```

12 – 23 – w założeniu ogólnym, aplikacja wykorzystuje układ grid o 3 wierszach i 3 kolumnach

TkInter 08 – przeglądarka zdjęć cz. 1.3

12, 13 – labelka umieszczona na samym dole okna, będzie pełnić rolę paska statusu, aby to pokreślić będzie nieco „wgłębiona” względem pozostałych elementów interfejsu (***relief = SUNKEN***). Będzie umieszczona w trzecim wierszu grid, rozciągnięta na 3 kolumnach, a jej zawartość będzie wyrównana do prawej (anchor na wschód). Cała kontrolka będzie rozciągnięta (sticky W+E)
15, 16 – podobnie jak poprzednio, wykorzystujemy tutaj obiekt klasy Label do wyświetlenie obrazu – domyślnie wyświetlamy tutaj pierwszą pozycję z listy obrazów (***self.listaObrazow[0]***). Obraz będzie wyświetlony w pierwszym wierszu i rozciągnięty na 3 kolumny
18 – 23 – poniżej obrazu wyświetlane są przyciski umożliwiające nawigację po liście obrazów (***przyciskWstecz***, ***przyciskNastepny***) oraz przycisk kończący prace programu (***przyciskKoniec***)
24 – niezależnie od wszystkiego wywołujemy metodę, która ustala wygląd aplikacji w danym momencie (***self.wyswietlaAktualny()***)

```
1   from tkinter import *
2   from PIL import ImageTk, Image
3   class TPrzegladarka:
4       def __init__(self, oknoRodzica,nazwyObrazow):
5           self.oknoRodzica = oknoRodzica
6           self.aktFoto=0
7           self.listaNazwObrazow=nazwyObrazow
8           self.listaObrazow=[]
9           for nazwa in self.listaNazwObrazow:
10              self.listaObrazow.append(ImageTk.PhotoImage(Image.open("zdjecia/"+nazwa)))
11
12          self.labelStatus = Label(oknoRodzica, text='testowa', bd=1,  relief=SUNKEN, anchor=E)
13          self.labelStatus.grid(row=2, column=0, columnspan=3, sticky=W+E)
14
15          self.labelWyswObraz = Label(image=self.listaObrazow[0])
16          self.labelWyswObraz.grid(row=0, column=0, columnspan=3)
17
18          self.przyciskWstecz = Button(self.oknoRodzica, text="Poprzedni", command=self.wyswietlPoprzedni, state=NORMAL)
19          self.przyciskWstecz.grid(row=1, column=0)
20          self.przyciskKoniec = Button(self.oknoRodzica, text="Wyjście z programu",  command = self.oknoRodzica.quit)
21          self.przyciskKoniec.grid(row=1, column=1)
22          self.przyciskNastepny = Button(self.oknoRodzica, text="Nastepny", command=self.wyswietlNastepny)
23          self.przyciskNastepny.grid(row=1, column=2)
24          self.wyswietlAktualny()
```

TkInter 08 – przeglądarka zdjęć cz. 2.1

```
25
26      def wyswietlNastepny(self):
27          if self.aktFoto<len(self.listaObrazow)-1:
28              self.aktFoto=self.aktFoto+1
29          self.wyswietlAktualny()
30
31      def wyswietlPoprzedni(self):
32          if self.aktFoto>0:
33              self.aktFoto=self.aktFoto-1
34          self.wyswietlAktualny()
35
36      def wyswietlAktualny(self):
37          self.labelWyswObraz['image']=self.listaObrazow[self.aktFoto]
38          self.labelStatus['text']='Zdjęcie '+str(self.aktFoto+1)+' z '+str(len(self.listaObrazow))
39          self.wyswietlPrzyciskPoprzedni()
40          self.wyswietlPrzyciskNastepny()
41
42      def wyswietlPrzyciskPoprzedni(self):
43          if self.aktFoto>0:
44              self.przyciskWstecz['state']=NORMAL
45          else:
46              self.przyciskWstecz['state']=DISABLED
47
48      def wyswietlPrzyciskNastepny(self):
49          if self.aktFoto<len(self.listaObrazow)-1:
50              self.przyciskNastepny['state']=NORMAL
51          else:
52              self.przyciskNastepny['state']=DISABLED
53
```

26 – metoda obsługujące kliknięcie w przycisk „Następny”
27 – sprawdzamy, czy numer aktualnie wyświetlanego zdjęcia jest mniejszy od ilości elementów na liście obrazów pomniejszonej o 1 (lista indeksowana jest od 0) – oznacza to w praktyce, że w danym momencie nie wyświetlamy jeszcze ostatniego obrazu
28 – jeśli warunek z ***27*** jest spełniony to zwiększamy numer zdjęcia, które powinno być wyświetlone.
29 – wyświetlamy aktualny stan aplikacji

31 - 34 – metoda analogiczna jak w przypadku poprzednim, tylko weryfikuje, czy możliwe jest przejście na jakieś zdjęcie poprzednie czy „jesteśmy” w danym momencie na pierwszym zdjęciu.

TkInter 08 – przeglądarka zdjęć cz. 2.2

```
25
26      def wyswietlNastepny(self):
27          if self.aktFoto<len(self.listaObrazow)-1:
28              self.aktFoto=self.aktFoto+1
29          self.wyswietlAktualny()
30
31      def wyswietlPoprzedni(self):
32          if self.aktFoto>0:
33              self.aktFoto=self.aktFoto-1
34          self.wyswietlAktualny()
35
36      def wyswietlAktualny(self):
37          self.labelWyswObraz['image']=self.listaObrazow[self.aktFoto]
38          self.labelStatus['text']='Zdjęcie '+str(self.aktFoto+1)+' z '+str(len(self.listaObrazow))
39          self.wyswietlPrzyciskPoprzedni()
40          self.wyswietlPrzyciskNastepny()
41
42      def wyswietlPrzyciskPoprzedni(self):
43          if self.aktFoto>0:
44              self.przyciskWstecz['state']=NORMAL
45          else:
46              self.przyciskWstecz['state']=DISABLED
47
48      def wyswietlPrzyciskNastepny(self):
49          if self.aktFoto<len(self.listaObrazow)-1:
50              self.przyciskNastepny['state']=NORMAL
51          else:
52              self.przyciskNastepny['state']=DISABLED
53
```

36 – metoda aktualizująca wygląd aplikacji zgodnie z aktualnym stanem
37 – wyświetlamy obraz o numerze wskazywanym przez ***self.aktFoto***
38 – ustalamy aktualny tekst wyświetlany w pasku statusu
39, 40 – wywołujemy metody ustalające aktualny wygląd przycisków sterujących

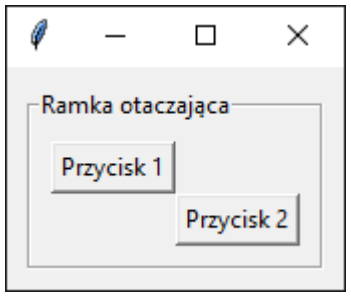
42 – metoda „sterująca” statusem przycisku „poprzedni”
43 – 46 – sprawdzamy, czy czasami numer aktualnie wyświetlanego obrazu nie wskazuje na pierwszy obraz z listy – w takiej sytuacji przejście na element poprzedni nie ma sensu i przycisk powinien być nie aktywny (***przyciskWstecz['state'] = DISABLED***). W przeciwnym wypadku oczywiście efekt powinien być odwrotny
48 – 52 – metod analogiczna jak wyżej tylko „obsługująca” przyciskNatepny

TkInter 08 – przeglądarka zdjęć cz. 3

```
57      okno = Tk()
58      okno.title("Przeglądarka zdjęć")
59      nazwyObrazow=['auto_1.jpg','auto_2.jpg','auto_3.jpg','auto_4.jpg','auto_5.jpg']
60      przegladarka = TPrzegladarka(okno,nazwyObrazow)
61      okno.mainloop()
```

59 – lista nazw plików obrazów, która będą wyświetlane
60 – tworzymy nowy obiekt klasy ***TPrzegladarka***, jako parametry konstruktora przekazujemy mu okno rodzica (***okno***) oraz przygotowaną w ***59*** listę obrazów

TkInter 09 – group box




```
1   from tkinter import *
2   okno = Tk()
3   okno.title('Program')
4   ramka = LabelFrame(okno, text='Ramka otaczająca', padx=10, pady=10)
5   ramka.pack(padx=10, pady=10)
6   przycisk1 = Button(ramka, text="Przycisk 1")
7   przycisk2 = Button(ramka, text="Przycisk 2")
8   przycisk1.grid(row=0, column=0)
9   przycisk2.grid(row=1, column=1)
10  okno.mainloop()
```

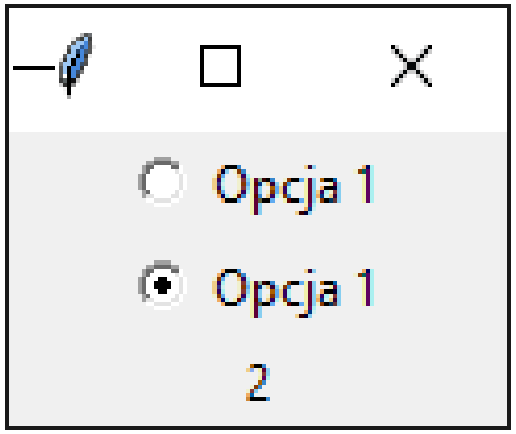
Group box lub inaczej ramka grupująca (otaczająca) to często wykorzystywany element GUI. Z jednej strony pozwala wizualnie odseparować fragment okna (dodając mu jeśli trzeba dodatkowy nagłówek), z drugiej tworzy nowy obszar z własnymi organizatorami układu.

4, 5 – obiekt klasy ***LabelFrame***. Jego rodzicem jest obiekt okno, ma ustalony tekst do wyświetlenia, oraz wielkość marginesów. Zostaje również osadzony w oknie z marginesami o wartości 10.

6, 7 – tworzymy dwa przyciski, w obu przypadkach jednak rodzicem nie jest okno o ramka przygotowana w 4.

8, 9 – do ułożenia i wyświetlenia przycisków używamy grid, przy czym nie jest to grid okna tylko grid ramki

TkInter 10 – grupa radiowa cz. 1




```
1   from tkinter import *
2
3   okno=Tk()
4   okno.title("Grupa radiowa")
5   wybor = IntVar()
6   wybor.set(2)
7   def wybranoOpcje(wartosc):
8       labelWynik['text']=wartosc
9
10  Radiobutton(okno, text="Opcja 1", variable=wybor, value=1, command=lambda: wybranoOpcje(wybor.get())).pack()
11  Radiobutton(okno, text="Opcja 1", variable=wybor, value=2, command=lambda: wybranoOpcje(wybor.get())).pack()
12
13  labelWynik = Label(okno, text=wybor.get())
14  labelWynik.pack()
15  okno.mainloop()
```

Tzw. grupa radiowa to również często wykorzystywany element interfejsu użytkownika – pozwala wygodnie wskazać wybrany przez niego element, przy czym zakłada się, że wybrana może być zawsze tylko jedna z opcji (wybór większej liczby jest technicznie nie możliwy).

5 – tkinter zawiera definicje klas odpowiadającym standardowym typom pytona, wyposażonym jednak w dodatkowe możliwości. Jedną z nich jest możliwość automatycznego łączenia ich (bindowania) z atrybutami innych widgetów tkintera. W tym przypadku tworzymy nowy obiekt klasy ***IntVar*** (odpowiednik typu całkowitego) pod nazwą ***wybor***. Będzie on miał zasięg skryptu

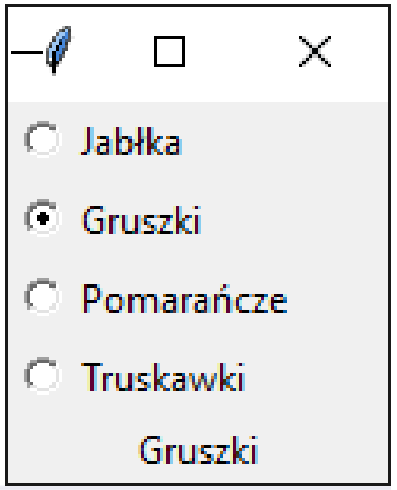
6 – wszystkie „tkinterowe” zmienne są wyposażone w metody ***get()*** oraz ***set()*** – wykorzystujemy ***set()*** aby ustawić jej domyślną wartość na 2.

TkInter 10 – grupa radiowa cz. 2

```
1   from tkinter import *
2
3   okno=Tk()
4   okno.title("Grupa radiowa")
5   wybor = IntVar()
6   wybor.set(2)
7   def wybranoOpcje(wartosc):
8       labelWynik['text']=wartosc
9
10  Radiobutton(okno, text="Opcja 1", variable=wybor, value=1, command=lambda: wybranoOpcje(wybor.get())).pack()
11  Radiobutton(okno, text="Opcja 1", variable=wybor, value=2, command=lambda: wybranoOpcje(wybor.get())).pack()
12
13  labelWynik = Label(okno, text=wybor.get())
14  labelWynik.pack()
15  okno.mainloop()
```

7, 8 – prosta metoda wyświetlająca wartość podaną jako parametr w pomocniczej labelce
10, 11 – dwa przyciski tworzące grupę radiową (obiekty klasy ***Radiobutton***) – w tym wypadku oba anonimowe. Skąd widomo, że to jedna grupa radiowa? Mają „podpiętą” tę samą zmienną (***variable = wybor***). Jako metodę obsługi podpięto w tym przypadku ***wybranoOpcje()*** z odpowiednim parametrem. Wartość tego parametru pobieramy z wybor.get(). Warto zwrócić uwagę na fakt, że nigdzie tego wyboru nie ustalaliśmy – jego wartość jest ustalana automatycznie poprzez połączenie z ***value***. Oba obiekty są anonimowe, zatem ***pack()*** zostało wywołana w tym samym wierszu, tuż po ich utworzeniu

TkInter 11 – grupa radiowa dynamiczna




```
1   from tkinter import *
2
3   okno=Tk()
4   okno.title("Grupa radiowa")
5
6   WYBORY=[
7       ("Jabłka","Jabłka"),
8       ("Gruszki","Gruszki"),
9       ("Pomarańcze","Pomarańcze"),
10      ("Truskawki","Truskawki")
11      ]
12
13  wybraneOwoce=StringVar()
14  wybraneOwoce.set("Jabłka")
15
16  for opis,owoce in WYBORY:
17      Radiobutton(okno, text=opis, variable=wybraneOwoce, value=owoce, command=lambda: wybranoOpcje(wybraneOwoce.get())).pack(anchor=W)
18
19  def wybranoOpcje(wartosc):
20      labelWynik['text']=wartosc
21
22  labelWynik = Label(okno, text=wybraneOwoce.get())
23  labelWynik.pack()
24  okno.mainloop()
```

Nieco zmodyfikowany przykład poprzedni – tym razem zawartość grupy radiowej będzie budowana na podstawie wcześniej przygotowanej listy.
6 – lista krotek, które staną się opcjami grupy radiowej (w tym wypadku wartości i wyświetlane napisy mają takie same wartości, co powoduje, że sens użycia krotek jest mniejszy niż pojedynczych napisów, ale gdy wartości i ich opisy były różne to było by to bardzo zasadne)

13, 14 – zmienna globalna tkintera – tym razem odpowiednik napisu
16, 17 – pętla w której przeglądamy krotki w liście ***WYBORY*** i na ich podstawie tworzymy nowe opcje dla grupy radiowej (rozgraniczamy tutaj opis oraz wartość)
19 – metoda wyświetlająca dokonany wybór