Informatyka II

Moduły i biblioteki w języku Python

Cel

Celem ćwiczenia jest zapoznanie się z pojęciem bibliotek i modułów w języku Python na przykładzie biblioteki guizero.

Zadanie 1

Uruchom środowisko JetBrains PyCharm. Możesz również skorzystać z własnego komputera lub z innych środowisk obecnych na komputerach w laboratorium. Upewnij się, że Twoje środowisko obsługuje język Python w wersji 3.6.1 lub nowszej. Stwórz nowy projekt

Krok 1. W oknie startowym wybierz opcję "Create New Project"



Krok 2. Wybierz miejsce zapisu i nazwę projektu

🖻 New Project	-		×
Location:			
Project Interpreter: New Virtualenv environment			
Project name can't be empty			
	Create	Ca	ncel

Krok 3. Gdy środowisko skończy tworzenie projektu, ukaże Ci się okno edytora

🖺 guizeroInstrukcja [] - PyCharm	- 🗆 X
<u>File Edit View N</u> avigate <u>C</u> ode <u>R</u> efactor R <u>u</u> n <u>T</u> ools VC <u>S</u>	<u>W</u> indow <u>H</u> elp	
🖿 guizeroInstrukcja	Ad	dd Configuration 🕨 🇯 🔳 🔍
명 🖻 Project 👻 😳 😤 🗢		
문 > IIII External Libraries		
Scratches and Consoles	Search Everywhere Double Shift	
Favorites	Go to File Ctrl+Shift+N	
≈i ★	Recent Files Ctrl+E	
Idure	Navigation Bar Alt+Home	
2. Str	Drop files here to open	
Python Console 🗵 Terminal 🔠 <u>6</u> : TODO		C Event Log
		° ⊨ ≣

Krok 4. Kliknij prawym przyciskiem myszy na nazwę projektu. Ukaże Ci się menu kontekstowe. Najedź kursorem myszy na pozycję "New", po czym wybierz opcję "Python File"

h	gu	izero	Inst	rukcja						
ţ		Pro	ject	•	0 I	\$	—			
<u>1</u> : Proj	>	g	•	New		··· · · · · · · · · · · · · · · · · ·	≝ Fi	le		1
			Ж	Cut	(Ctrl+X	🖹 N	ew Scratch File	Ctrl+Alt+Shift+Insert	
		-03	ē	<u>C</u> opy	(Ctrl+C	D	irectory		
				C <u>o</u> py Path	Ctrl+Sł	nift+C	P P	ython Package		
				Copy Relative Path	Ctrl+Alt+Sł	hift+C	🐌 Р	ython File		
			Ů	<u>P</u> aste	(Ctrl+V	ill H	TML File		
				Find <u>U</u> sages	A	Alt+F7	📊 R	esource Bundle		
				Find in <u>P</u> ath	Ctrl+SI	hift+F		Search	Everywhere Double S	Shii

Krok 5. Nadaj plikowi nazwę (bez rozszerzenia!), po czym wciśnij przycisk "OK". Plik powinien zostać automatycznie otwarty do edycji

🖻 New Python file											
<u>N</u> ame:	main		ţ↑								
<u>K</u> ind:	층 Python file		\sim								
		OK Can	cel								

Zadanie 2

Zaznajom się z formą dokumentacji biblioteki *guizero* (<u>https://lawsie.github.io/guizero/about</u>). Znajdziesz w niej szczegółowe informacje o wszystkich klasach i metodach jakie oferuje biblioteka, a także wskazówki dotyczące instalacji. Jeżeli korzystasz ze środowiska wskazanego przez prowadzącego zajęcia, konieczne jest dodanie biblioteki do środowiska projektu. 1. Z paska menu wybierz pozycję "File", a następnie "Settings"



 W oknie ustawień rozwiń zakładkę "Project: nazwa projektu", po czym wybierz pozycję "Project Interpreter". W oknie po prawej powinny pokazać się dwie pozycje. Dwukrotnie kliknij na pozycję "pip".

PC Settings					×
Qv		Project: guizerolns	strukcja > Projec	t Interpreter	ē
✓ Appearance & Behavior		Project Interpreter:	📌 Python 3.7 (g	uizerolnstrukcja) 🗠	\$
Appearance					
Menus and Toolbars		Package	Version	Latest version	+
System Settings		pip	19.0.3	▲ 19.1.1	_
File Colors	ē	setuptools	40.8.0	41.0.1	
Scopes	ē				0
Notifications					
Quick Lists					
Кеутар					
> Editor					
Plugins					
> Version Control	ē				
 Project: guizeroInstrukcja 	ē				
Project Interpreter	œ.				
Project Structure	ē				
> Build, Execution, Deployment					
> Languages & Frameworks					
> Tools					
0			OK	Cancel App	ly

3. W oknie wyszukiwania wpisz "guizero", po czym wciśnij przycisk "Install Package" i poczekaj aż środowisko zainstaluje pakiet.

PC Available Packages				×
Q∗guizero				×
guizero	ទ	Description		
		Python module to	allow learners to easily create GUIs	
		Version		
		0.6.4		
		Author		
		Laura Sach		
		mailto:laura.sach(https://github.com	@raspberrypi.org /lawsie/guizero	
		Specify version	0.6.4	~
		Options		
Install Package	age Repositories			

4. Po zainstalowaniu biblioteki możesz zamknąć okno pakietów oraz okno ustawień i wrócić do okna edytora.

Zadanie 3

Guizero to biblioteka służąca do projektowania aplikacji okienkowych. Naszym celem będzie stworzenie gry wzorowanej na grze *Saper* (dostępnej we wcześniejszych edycjach systemu Windows).

Rozpocznijmy pracę od wyświetlenia okna naszej aplikacji. Przepisz następujący kod, a następnie uruchom projekt:

from guizero import *

```
gra = App("Saper",450,520)
gra.display()
```

Krok 1. Z paska menu wybierz pozycję "Run", następnie z menu rozwijanego "Run..". W wyskakującym oknie wybierz swój plik z kodem

PC	guizeroInstru	ikcja []]\main.py	y [guizero	olnstrukcja]	- PyCharr	m							-			×
<u>F</u> ile	<u>E</u> dit <u>V</u> iew	<u>N</u> avigate	<u>C</u> ode	<u>R</u> efact	tor	R <u>u</u> n	<u>T</u> ools	VC <u>S</u>	<u>W</u> indow	v <u>F</u>	<u>H</u> elp														
	guizeroInstr	ukcja 👌 💑	main.p	у		► R	l <u>u</u> n					S	Shift+F10						Ad	d Confi	iguration		ă		Q
Ħ		- m	_	ila maiu	0.04	ĕ₫	<u>ebug</u>						Shift+F9									_			
je.	in the C	· · · · · ·	_			🕨 R						Alt+S	Shift+F10												_
- E	✓ guizer	olnstrukcja		1	fr	ð D	ebug					Alt+	Shift+F9												
	> ve	nv library ro	ot	2		т. А	ttach to	Proce	255			Ctr	I+Alt+F5												
	i 🧑 ma	iin.py		3	gr	₽ E	dit Conf	igurat	ions																-
e G	III Extern	al Libraries		5	91	<u> </u>	une com	-t D	- des																
lori	🔊 Scratcl	hes and Con	soles				троп те	st Kes	uits																
Fa						S	top						Ctrl+F2												
~						S	top Bacl	grou	nd Proces	ses.		Ctrl+	Shift+F2												
~						S	how Ru	nning	List																
ture						≙ s	tep <u>O</u> ve	r					F8												
Ĕ						△ F	orce Ste	p O <u>v</u> e	r			Alt+	Shift+F8												
Z: S)						<u>+</u> s	tep <u>I</u> nto						F7												
	Python Co	onsole 🗵	Termi	nal i≣	<u>6</u> : 1	<u></u>	orce Ste	p l <u>n</u> to				Alt+	Shift+F7										1 Ev	ent Lo	g
	Choose and	run configu	ration			±+ s	mart Ste	p Into					Shift+F7	2:1	1 n/	a UTF-	-8 ÷ 4	spaces	÷F	vthon	3.7 (quize	rolnstru	(kcia)	÷ 9	

Krok 2. Następnie projekt powinien się uruchomić, a Twoim oczom ukazać puste okno o nazwie "Saper" mające 450 pikseli szerokości i 520 pikseli wysokości



Krok 3. Od teraz możesz uruchamiać projekt przyciskiem "Run" (skrót klawiszowy: Shift+F10), który znajduje się w prawym-górnym obszarze okna środowiska

Aby zaimportować pewne elementy (klasy, metody) biblioteki, stosuje się konstrukcję *from [nazwa biblioteki] import [elementy]*. Jeżeli chcemy zaimportować wszystkie elementy z danej biblioteki, możemy listę elementów zamienić gwiazdką.

Klasa *App* służy do zarządzania interfejsem graficznym. Możesz o niej dokładniej poczytać w dokumentacji guizero: <u>https://lawsie.github.io/guizero/app</u>

Zadanie 4

Nasza gra będzie składać się z pola tekstowego (*TextBox*) w którym będziemy definiować ilość bomb, przycisku (*PushButton*) pozwalającego rozpocząć grę oraz planszy składającej się ze 100 przycisków, które będą symulować pola planszy. Plansze umieścimy w specjalnym kontenerze (*Box*), którego ustawimy w taki sposób, aby zawarte w nim elementy, rozmieszczone były w formie siatki (*"grid"*).

Pomiędzy stworzeniem aplikacji, a wywołaniem funkcji "display", umieść następujący kod:

ilośćBomb = TextBox(gra,"10",width="fill")
przyciskStart = PushButton(gra,text="Rozpocznij grę",width="fill")
plansza = Box(gra,"grid")

Jeżeli wszystko zostało wykonane poprawnie, okno powinno wyglądać w następujący sposób:



Zadanie 5

Zauważ, że po wciśnięciu przycisku nic się nie dzieje. Aby to zmienić należy zdefiniować funkcję obsługi kliknięcia i sprawić, żeby była wywoływana w reakcji na wciśnięcie przycisku. Zacznijmy od uproszczonej funkcji obsługi – po wciśnięciu przycisku wypiszemy w konsoli wiadomość.

W tym celu stworzymy następującą funkcję, której definicję umieścimy zaraz po importowaniu biblioteki, przed stworzeniem zmiennej aplikacji:

```
def rozpocznijGrę():
    print("Rozpoczynamy grę w sapera!")
```

Zmienimy też definicję przycisku start na następującą:

przyciskStart = PushButton(gra,rozpocznijGrę,text="Rozpocznij grę",width="fill")

Jeżeli wszystko zostało wykonane poprawnie, po wciśnięciu przycisku, w konsoli powinien zostać wyświetlony odpowiedni komunikat:



Zadanie 6

Zajmijmy się teraz tworzeniem planszy do gry. W tym celu potrzebujemy zrobić kilka rzeczy. Po pierwsze, stwórzmy klasę, która będzie przechowywała informacje o pojedynczym polu gry:

```
class Pole:
wizualizacja = None
maBombę = False
```

Jej definicję umieśćmy przed definicją funkcji *rozpocznijGrę*. Następnie dobrze byłoby wygenerować listę indeksów pól, które będą zawierały bomby. W tym celu zaimportujemy moduł random, który jest jednym z bazowych modułów języka. Następującą instrukcję należy umieścić tuż pod importem biblioteki guizero:

import random

Teraz w funkcji *rozpocznijGrę* możemy wygenerować listę pól z bombami. W tym celu skorzystamy z metody *sample* (więcej informacji możesz znaleźć w dokumentacji Pythona: <u>https://docs.python.org/3/library/random.html</u>). Aby pobrać wartość z elementy typu *TextBox*, można użyć funkcji *get* (<u>https://lawsie.github.io/guizero/textbox</u>).

```
random.seed()
global ilośćBomb
listaPólZBombami = random.sample(range(100), int(ilośćBomb.get()))
```

Dodatkowo, tuż po zdefiniowaniu zmiennej *plansza*, stworzymy zmienną przechowującą pola jako takie i przypiszemy do niej pustą listę.

```
pola = []
```

Zadanie 7

Zmodyfikujmy teraz funkcję *rozpocznijGrę* w taki sposób, aby po jej wywołaniu, tworzone były przyciski odpowiadające polom gry. W tym celu wykorzystamy następujący kod, który dopiszemy bezpośrednio w ciele funkcji:

```
global pola
 for y in range(10):
   for x in range(10):
      # Stwórz nowe pole gry
      pole = Pole()
      # Oblicz jego identyfikator
     idPola = x+y*10
      # Stwórz wizualizację
      pole.wizualizacja = PushButton(plansza,text="",grid=[x,y+1],width=1,height=1)
      # Sprawdź czy pole ma zawierać bombę
     if idPola in listaPólZBombami:
       pole.maBombe = True
      else:
        pole.maBombę = False
      # Dodaj pole do listy pól
      pola.append(pole)
```

Jeżeli wszystko zostało wykonane poprawnie, po wciśnięciu przycisku *rozpocznijGrę*, okno gry powinno przybrać wygląd zaprezentowany po prawej stronie.

Jeżeli coś nie działa, zatrzymaj się i popraw błędy oraz upewnij się, czy każdy fragment kodu jest umieszczony w poprawnym miejscu. Jeżeli nie jesteś w stanie wykonać tego samodzielnie, poproś o pomoc.

Ø	Saper						-	×
D								
		 	F	Rozpoc	znij gr	ę	 	

Zadanie 8

Ostatnią rzeczą jaką należy zrobić jest oprogramowanie reakcji pola na kliknięcie. W tym celu, podobnie jak w zadaniu 5, musimy zadeklarować funkcję obsługi zdarzenia oraz podpiąć ją do przycisku reprezentującego pole.

Istnieje jeden dodatkowy problem: każdy przycisk powinien reagować inaczej, w zależności od tego, czy pole zawiera bombę, czy też nie. Aby osiągnąć taki stan, należałoby do funkcji dodatkowo przekazać informację o tym, które pole zostało aktualnie wciśnięte.

Biblioteka guizero pozwala podczas obsługi zdarzeń przesłać do nich argumenty. Aby nie ograniczać użytkownika biblioteki, argumenty przekazujemy w formie listy i odbieramy w specjalny sposób, definiując każdą funkcję obsługi jako funkcję przyjmującą *dowolną* ilość argumentów. W języku Python należy w tym celu przekazać do funkcji *jeden* argument oraz poprzedzić go znakiem "*". Następnie argument można interpretować jako listę.

W analizowanym przypadku, zdefiniujemy następującą funkcję obsługi kliknięcia:

```
def kliknięcie(*listaArgumentów):
```

```
# Pobierz pozycję w poziomie pola z listy argumentów
x = listaArgumentów[0]
# Pobierz pozycję w pionie pola z listy argumentów
y = listaArgumentów[1]
```

```
# Pobierz pole z tablicy z polami
global pola
pole = pola[x+y*10]
```

```
# Jeżeli pole ma bombę, zniszcz wszystkie pola i wyczyść tablicę
if pole.maBombę:
    for pole in pola:
        pole.wizualizacja.destroy()
        pola.clear()
# Jeżeli pole nie ma bomby, zaznacz go znakiem "X"
else:
    pole.wizualizacja.text = "X"
```

Musimy jeszcze tylko podpiąć funkcję do przycisku i przekazać argumenty (pozycję w poziomie i pionie). Aby to zrobić, należy zmodyfikować definicję wizualizacji, która znajduje się w funkcji *rozpocznijGrę*:

```
pole.wizualizacja = PushButton(plansza,kliknięcie,(x,y),"",grid=[x,y+1],width=1,height=1)
```

Zadanie 9

Zmień logikę gry w taki sposób, by po kliknięciu pola gry *niezawierającego* bomby, zamiast zwykłego symbolu "X", gra wyświetliła ilość bomb, które znajdują się w sąsiedztwie klikniętego pola.

Zadanie 10

Zmień logikę gry w taki sposób, by po kliknięciu pola gry *niezawierającego* bomby, przycisk był kolorowany w zależności od ilości sąsiadujących z nim pól z bombami. Jeżeli wokół pola znajdują się cztery bomby, powinno przyjąć kolor czerwony. Trzy bomby: pomarańczowy, dwie bomby: żółty, jedną bombę: zielony, a żadnej bomby: biały.

Zadanie 11

Zmień logikę gry w taki sposób, by po kliknięciu pola gry zawierającego bombę, wszystkie przyciski zostały zdezaktywowane. Przyciski zawierające bombę powinny zmienić kolor na czerwony, a reszta przycisków powinna zmienić kolor tła na zielony.

Zadanie 12

Dodaj pod plansza tekst informujący o tym, ile pól odkrył gracz w danej rozgrywce. Gdy gracz odkryje wszystkie pola bez bomb, pola zawierające bomby powinny zostać deaktywowane, a ich tło ustawione na czerwone. Pamiętaj o resetowaniu licznika po rozpoczęciu nowej gry.

Zadanie 13

Zmień logikę gry w taki sposób, aby gracz nie mógł zginąć przy kliknięciu pierwszego pola. Przetestuj swoje rozwiązanie na planszy zawierającej 99 bomb.