INFORMATYKA II

JĘZYK PYTHON W PRAKTYCE

CEL LABORATORIUM

Celem zajęć jest przypomnienie podstawowych założeń języka Python i zapoznanie uczestniczki/uczestnika z bardziej zaawansowanymi możliwościami oferowanymi przez język. Podczas zajęć przedstawiona zostanie konstrukcja programu składającego się z wielu plików, korzystającego z dodatkowych modułów oraz zaprojektowanego w myśl paradygmatu programowania obiektowego.

WAŻNE INFORMACJE O ZAJĘCIACH

Niniejsza instrukcja oraz instrukcje przedstawione na kolejnych zajęciach zawierają kilka elementów pozwalających lepiej zorientować się w treści tychże.

- Sekcje oznaczone na **CZERWONO** (tak jak ta) są **ważne** i nie powinny zostać pominięte
- Sekcje oznaczone na CZARNO (takie jak sekcja „CEL LABORATORIUM" powyżej) pełnią funkcję informacyjną i mogą zostać pominięte, **jeżeli** uczestniczka/uczestnik zajęć nie uzna ich za interesujące.
- Sekcje oznaczone na **NIEBIESKO** (takie jak sekcja „KOMUNIKACJA Z BAZĄ DANYCH ZA POMOCĄ PAKIETU MYSQL.CONNECTOR") zawierają materiały dodatkowe, których umieszczenie w niniejszej instrukcji było niemożliwe lub niepraktyczne. Mogą zostać pominięte, lecz zapoznanie się z nimi może znacznie przyspieszyć pracę.

Z treścią instrukcji (jeżeli jest dostępna) należy zapoznać się najpóźniej w **dniu poprzedzającym** zajęcia. Podczas zajęć mogą zostać zadane pytania (zarówno w formie ustnej, jak i pisemnej), których celem będzie stwierdzenie stopnia przygotowania uczestników do zajęć.

Podczas zajęć uczestniczka/uczestnik powinien wykonać **obowiązkowe** zadania znajdujące się w instrukcji. Niewykonanie zadań podczas trwania laboratorium może skutkować **oceną negatywną**.

W instrukcji znajdują się również zadania dodatkowe, których wykonanie podczas zajęć jest **nieobowiązkowe**. Są one oznaczone symbolem "Ⓟ". Pierwsza uczestniczka/pierwszy uczestnik zajęć, który zgłosi wykonanie takiego zadania, otrzyma plusa. Ilość plusów jakie można zdobyć podczas zajęć jest nieograniczona. Zadania wykonane przed rozpoczęciem zajęć nie będą nagradzane, chyba że nikt inny nie zgłosi ich wykonania. W przypadku konfliktów, nagrodzona zostanie osoba z mniejszą ilością plusów.

Jeżeli uczestniczka/uczestnik natrafi na problem, powinien spróbować go rozwiązać samodzielnie (sięgając po informacje zawarte na Internecie lub bazując na własnej wiedzy i doświadczeniu). Jeżeli rozwiązanie nie zostanie znalezione w sensownym[1] czasie, należy skonsultować się z innymi, pobliskimi uczestnikami, a w razie dalszego braku rozwiązania,

[1] Za sensowny czas uznaje się czas nie krótszy niż 5 minut i proporcjonalny do problemu oraz pozostałej do realizacji ilości materiału.

z osobą prowadzącą zajęcia. Brak zrozumienia nie jest jednoznaczny z brakiem przygotowania (chyba, że bezpośrednio z niego wynika).

Powyższe informacje dotyczą **wszystkich** zajęć laboratoryjnych i instrukcji. **Nie będą one powtarzane** w kolejnych instrukcjach.

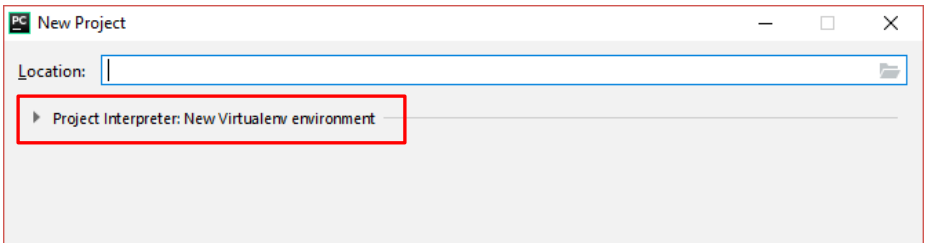
TWORZENIE NOWEGO PROJEKTU W ŚRODOWISKU PYCHARM

Podana w tym rozdziale procedura jest poprawna dla komputerów w laboratorium. Na innych stanowiskach i wersjach programu PyCharm, mogą wystąpić różnice.

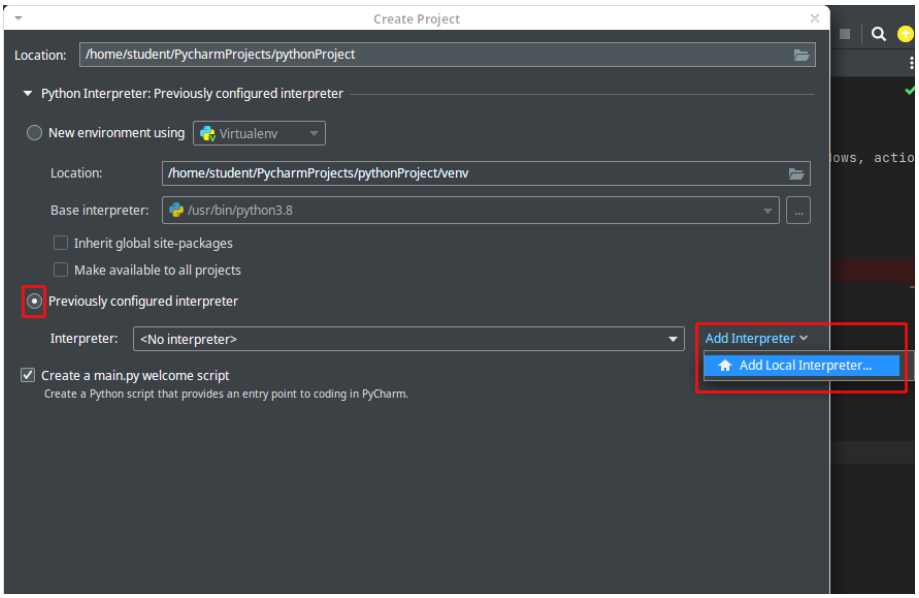
Uruchom środowisko JetBrains **PyCharm**. Możesz również skorzystać z własnego komputera lub z innych środowisk obecnych na komputerach w laboratorium. Upewnij się, że Twoje środowisko obsługuje język Python w wersji 3.6.1 lub nowszej. Stwórz nowy projekt.

Krok 1. Po uruchomieniu programu na komputerach w laboratorium automatycznie otworzy się projekt „first". Z menu głównego aplikacji wybierz pozycję „File", a następnie „New Project".

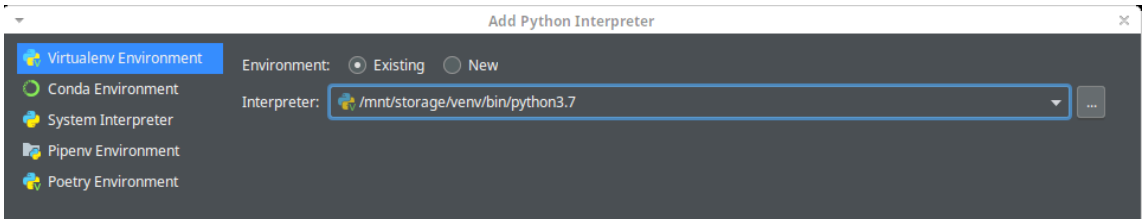
Krok 2. W nazwie zawrzyj swoje imię i nazwisko oraz aktualną datę, np. „Adam Mickiewicz 11 02 2025". Rozwiń ustawienia interpretera



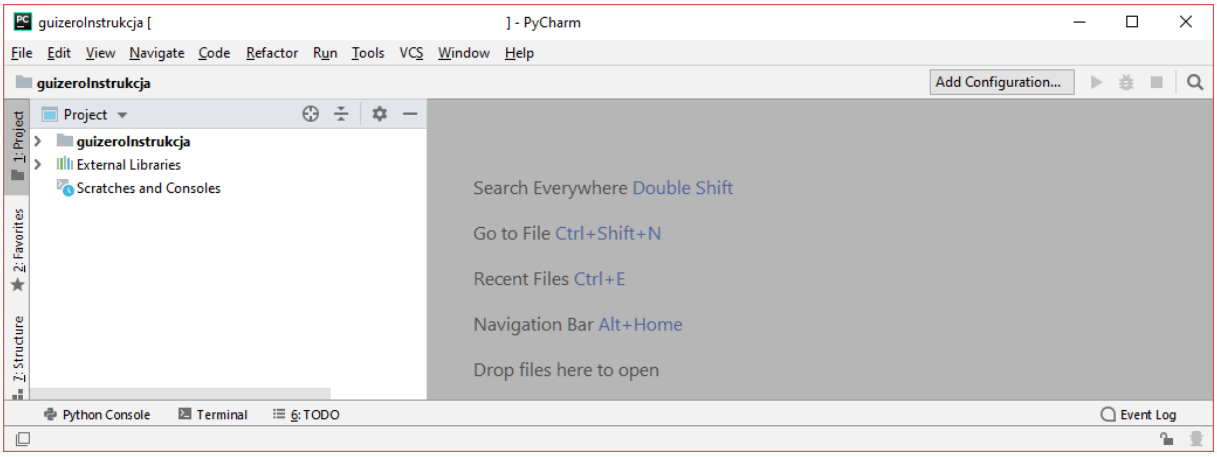
Krok 3. Zaznacz opcję „Previously configured interpreter". Następnie kliknij przycisk "Add Interpreter" i wybierz opcję „Add Local Interpreter".



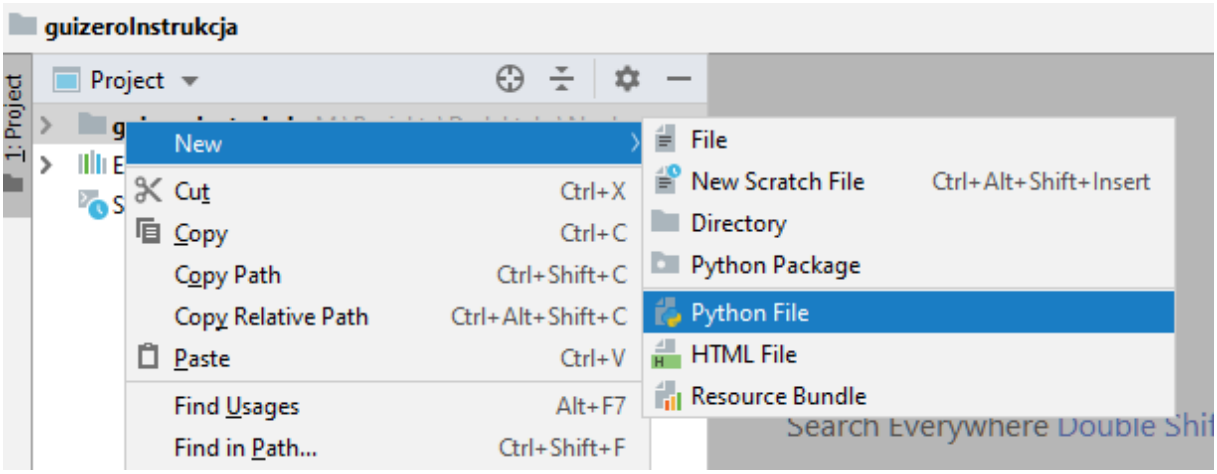
Krok 4. W oknie dodawania interpretera wybierz opcję „Existing", a w polu Interpreter wypisz „*/mnt/storage/venv/bin/python3.7*". Zamiast wpisywania możesz kliknąć przycisk „…" i wyszukać plik interpretera pod podaną ścieżką.



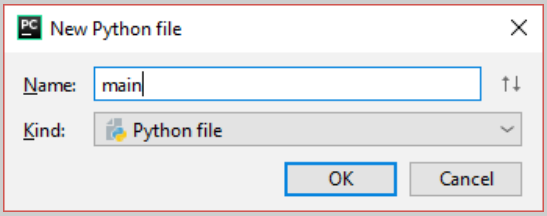
Krok 5. Kliknij przycisk „OK", aby zamknąć okno.
Krok 6. W oknie tworzenia projektu kliknij przycisk „Create".
Krok 7. Gdy środowisko skończy tworzenie projektu, ukaże Ci się okno edytora.



Krok 8. Kliknij prawym przyciskiem myszy na nazwę projektu. Ukaże Ci się menu kontekstowe. Najedź kursorem myszy na pozycję „New", po czym wybierz opcję „Python File"



Krok 9. Nadaj plikowi nazwę *main* (z rozszerzeniem *.py!), po czym wciśnij przycisk „OK". Plik powinien zostać automatycznie otwarty do edycji. Będzie to główny plik naszej aplikacji



SPECYFIKACJA APLIKACJI

Celem zajęć będzie stworzenie aplikacji konsolowej, która będzie służyć do zarządzania stanem hipotetycznego inteligentnego domu. Dom ten jest wyposażony w czujniki pozwalające stwierdzić, gdzie aktualnie są mieszkańcy, jakie urządzenia działają, jaka jest temperatura oraz kontrolery pozwalające włączyć telewizor, czajnik czy światło w salonie.

Informacje zwracane przez czujniki przechowywane będą w bazie danych (która zostanie przedstawiona na kolejnych zajęciach), zaś napisany w języku Python program będzie komunikował się z rzeczoną bazą danych za pomocą zapytań języka SQL (który zostanie lub został przedstawiony na wykładzie).

Program będzie się składał z kilku modułów (każdy moduł będzie osobną klasą), dzięki czemu modyfikowanie i rozszerzanie go w przyszłości będzie łatwe oraz możliwe do zrównoleglenia.

Centralnym plikiem programu będzie plik *main*, utworzony w poprzednim punkcie instrukcji. Oprócz tego utworzona zostanie klasa „`Aplikacja`", która będzie zarządzać działaniem całej aplikacji oraz klasa „`Moduł`", reprezentująca pewien abstrakcyjny fragment programu. Konkretne moduły będą dziedziczyć po klasie `Moduł`.

GŁÓWNY PLIK PROGRAMU

Działanie naszego programu można przedstawić w postaci następującej listy kroków:

1. Utwórz obiekt klasy Aplikacja
2. Zarejestruj wszystkie potrzebne moduły w klasie Aplikacja
3. Jeżeli Aplikacja działa, to:
4. Wyświetl menu
5. Poczekaj aż użytkownik dokona wyboru modułu
6. Przekaż kontrolę nad aplikacją do danego modułu
7. Wróć do kroku 3

Powyższy algorytm można zobrazować w postaci następującego kodu w języku Python:

```
# utwórz aplikację
aplikacja = Aplikacja()

# zarejestruj moduły
# …których na chwilę obecną nie mamy…

# dopóki aplikacja działa
while(aplikacja.działa):
    # wyświetl menu
    aplikacja.wyświetlMenu()
```

```
# pozwól użytkownikowi wybrać moduł i przekaż kontrolę
wybór = input("Wybierz moduł: ")
aplikacja.wybierzModuł(wybór)
```

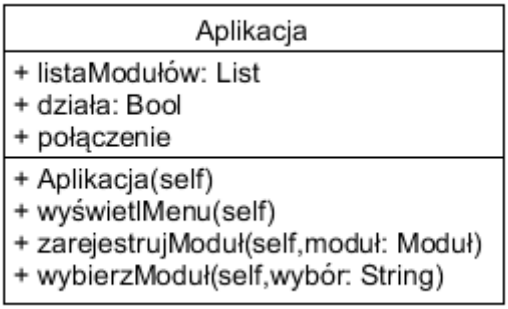
Przepisz powyższy kod do pliku *main*. Oczywiście programu nie można w tej chwili uruchomić, gdyż nie utworzyliśmy klasy `Aplikacja`. Zrozumienie powyższego kodu nie powinno przysparzać trudności. Jeżeli jest inaczej, sięgnij po notatki wykonane na zajęciach z przedmiotu Informatyka I.

OMÓWIENIE KLASY „APLIKACJA"

Z poprzedniego kodu wynika, że w klasie `Aplikacja` znaleźć się musi jakaś zmienna o nazwie „`działa`" oraz metody „`wyświetlMenu`" i „`wybierzModuł`".

Przydałaby się również metoda pozwalające dodać nowy moduł do aplikacji (nazwiemy ją „`zarejestrujModuł`"). Sama klasa musi przechowywać również listę zarejestrowanych modułów. W tym celu stworzymy w klasie zmienną „`listaModułów`".

Dodatkowo – nasza aplikacja będzie korzystała z bazy danych. Warto byłoby umieścić w klasie `Aplikacja` zmienną przechowującą połączenie z bazą danych. Dzięki temu, gdyby jakiś moduł potrzebował skorzystać z tego połączenia, będzie mógł w łatwy sposób uzyskać do niego dostęp. Zmienną przechowującą dostęp do bazy danych nazwiemy „`połączenie`". Połączenie z bazą nawiążemy w konstruktorze klasy.



W związku z tym, klasę `Aplikacja` można przedstawić za pomocą diagramu klasy przedstawionego powyżej.

KLASA „APLIKACJA"

Rozpocznijmy od utworzenia nowego pliku. Nazwijmy ten plik „aplikacja". Jeżeli nie pamiętasz, jak dodać plik do projektu, wróć do sekcji „TWORZENIE NOWEGO PROJEKTU W ŚRODOWISKU PYCHARM" (kroki 8 i 9).

Do nowoutworzonego pliku **przepisz** następujący kod:

```
class Aplikacja:
    listaModułów = []
    działa = True
    połączenie = None

    def __init__(self):
        # W tym miejscu znajdzie się kod inicjalizujący połączenie z bazą
        # danych

    def wyświetlMenu(self):
        # W tym miejscu znajdzie się kod wyświetlający nagłówek menu
        # oraz kod wypisujący listę modułów
```

```
    def zarejestrujModuł(self, moduł):
        # W tym miejscu dodamy moduł do listy listaModułów
        # oraz upewnimy się, że moduł wie o naszej aplikacji

    def wybierzModuł(self, wybór):
        # Jeżeli użytkownik wybrał moduł, tutaj przekażemy temu modułowi
        # kontrolę
```

Na chwilę obecną, konstruktor nie jest potrzebny, dlatego jego ciało będzie się składać z jednej instrukcji: `pass`.

```
def __init__(self):
    pass
```

Zacznijmy uzupełniać nasz kod od ciała metody `wyświetlMenu`. Algorytm jej działania można przedstawić w postaci następującej listy kroków:

1. Wypisz na ekranie nagłówek menu
2. Jeżeli a aplikacji nie zarejestrowano żadnych modułów, to:
   2.1. Wypisz na ekranie komunikat „Brak zarejestrowanych modułów!"
   2.2. Zakończ działanie aplikacji
3. Dla każdego zarejestrowanego modułu, powtórz:
   3.1. Wypisz identyfikator używany do aktywacji
   3.2. Wypisz nazwę modułu

Powyższy algorytm jest realizowany przez następujący kod:

```
def wyświetlMenu(self):
    print("\nLista modułów:")
    print("======================")
    if len(self.listaModułów)==0:
        print("Brak zarejestrowanych modułów!")
        self.działa = False

    for moduł in self.listaModułów:
        print("[", moduł.identyfikator, "]\t", moduł.nazwa)
```

W następnej kolejności zajmiemy się metodą `zarejestrujModuł`. Po jej wywołaniu po prostu dodamy przekazany jako drugi argument moduł do listy modułów, za pomocą funkcji `append`. Następnie przypiszemy polu *aplikacja* modułu wartość `self`. Służy do tego poniższy kod źródłowy:

```
def zarejestrujModuł(self, moduł):
    self.listaModułów.append(moduł)
    moduł.aplikacja = self
```

Ostatnią funkcją jest metoda `wybierzModuł`. Jej działanie opisuje następująca lista kroków:

1. Jeżeli wartość zmiennej wybór jest równa ciągowi znaków „koniec", to:
   1.1. Zakończ działanie aplikacji
2. Dla każdego zarejestrowanego modułu, powtórz:
   2.1. Jeżeli wartość zmiennej wybór jest równa identyfikatorowi modułu, to:
      2.1.1. Przekaż kontrolę nad aplikacją do modułu

Którą implementuje poniższy kod języka Python:

```
def wybierzModuł(self, wybór):
    if(wybór.upper() == "KONIEC"):
        self.działa = False
        return
    for moduł in self.listaModułów:
        if wybór.upper() == moduł.identyfikator.upper():
            moduł.przejmijKontrolę()
```

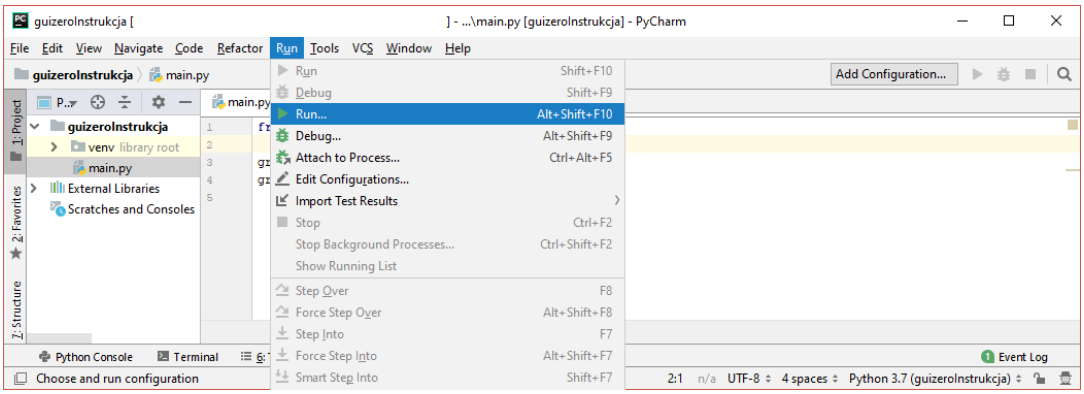
Funkcja `upper` dba o to, by identyfikatory były rozpoznawane niezależnie od wielkości liter. Aplikacja przestanie działać zarówno po wpisaniu słów „KONIEC", „Koniec", jak i „koniec".

Aby klasa `Aplikacja` była widoczna w pliku *main*, do pliku *main* musimy dodać instrukcję importującą zawartość pliku „*aplikacja*". W tym celu, na samej górze pliku, przed użyciem klasy `Aplikacja`, należy dodać następującą linijkę kodu:

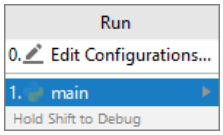
```
from aplikacja import *
```

Jeżeli (wbrew instrukcji) plik nazywa się inaczej, należy słowo „*aplikacja*" zamienić na nazwę pliku. Program można teraz uruchomić i przetestować. W tym celu wykonaj następujące kroki:

Krok 1. Z paska menu wybierz pozycję „Run", następnie z menu rozwijanego „Run..".



Krok 2. W okienku, które się pojawi na środku ekranu, wybierz plik **main**.



Po wykonaniu powyższych kroków, w konsoli znajdującej się w dolnej części interfejsu edytora, powinien się wyświetlić następujący tekst:

```
Lista modułów:
======================
Brak zarejestrowanych modułów!
Wybierz moduł:
```

KOMUNIKACJA Z BAZĄ DANYCH ZA POMOCĄ PAKIETU MYSQL-CONNECTOR

Używana na laboratorium baza danych działa w oparciu o system MySQL. Firma Oracle, która posiada prawa do systemu MySQL, opracowała pakiet pozwalający na komunikację z bazą MySQL z poziomu programu napisanego w języku Python.

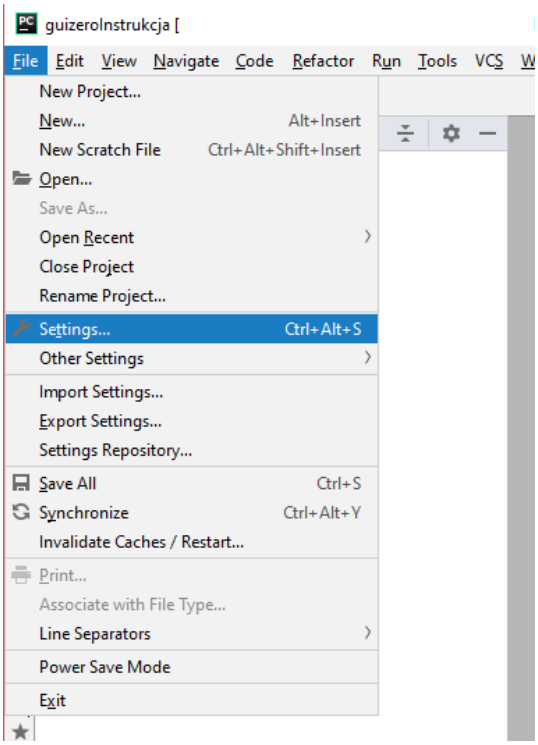
Dokumentację pakietu mysql-connector możesz znaleźć na stronie https://dev.mysql.com/doc/connector-python/en/

Dokumentację systemu MySQL możesz znaleźć na stronie https://dev.mysql.com/doc/refman/8.0/en/

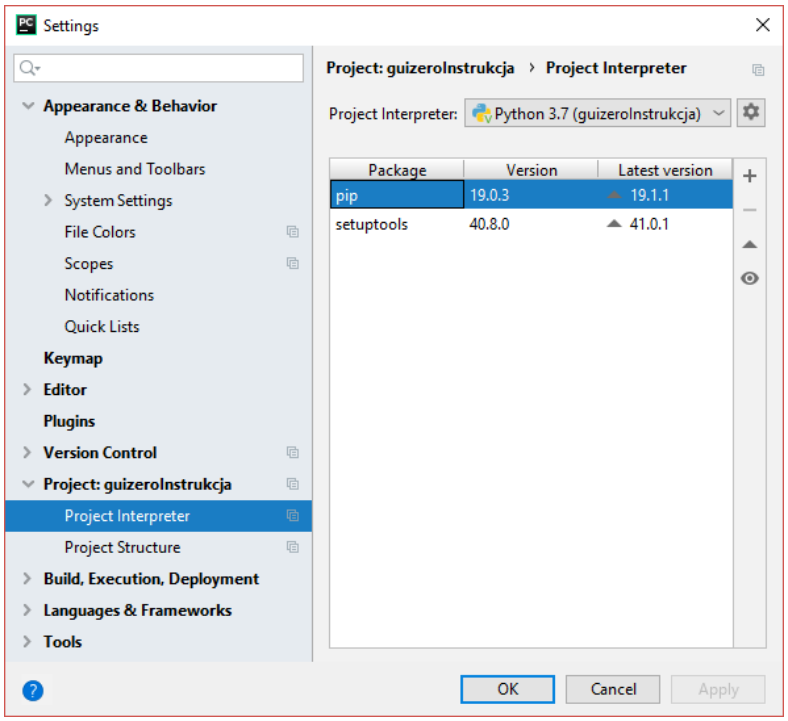
INSTALACJA PAKIETU MYSQL-CONNECTOR-PYTHON

Kolejnym krokiem jest integracja pakietu „mysql-connector-python" z projektem. **Jeżeli pracujesz na stanowisku laboratoryjnym to pomiń treść tego rozdziału**. Aby to zrobić, wykonaj następujące kroki:

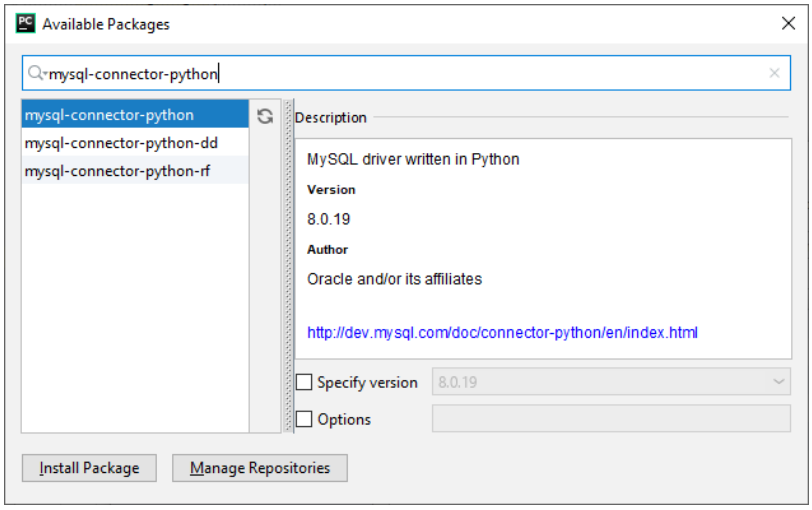
Krok 1. Z paska menu wybierz pozycję „File", a następnie „Settings"



Krok 2. W oknie ustawień rozwiń zakładkę „Project: nazwa projektu" (na poniższym rysunku projekt nazwano „guizerolnstrukcja"), po czym wybierz pozycję „Project Interpreter". W oknie po prawej powinny pokazać się dwie pozycje. Kliknij na ikonkę „+" znajdującą się po prawej stronie nagłówka tabeli.



Krok 3. W oknie wyszukiwania wpisz „mysql-connector-python", wybierz z listy pakiet „mysql-connector-python" (bez żadnych dopisków), po czym wciśnij przycisk „Install Package" i poczekaj aż środowisko zainstaluje pakiet. W oknie „Available Packages" wyświetli się informacja. Na komputerach o mniejszej mocy obliczeniowej, może to zająć do kilku minut.



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

UTWORZENIE POŁĄCZENIA Z BAZĄ DANYCH

Utwórz nowy plik i nazwij go „konfiguracja". Umieść w nim następujący kod:

```
użytkownik = '???'
hasło = '???'
adres = '???'
nazwaBazy = '???'
```

W miejscach oznaczonych znakami zapytania wpisz poprawne dane użytkownika, hasła i bazy. Dane udostępnia prowadzący. Spójrz na tablicę lub na kartkę, którą otrzymałeś w trakcie zajęć. Gdy uzupełnisz już dane, wróć do pliku zawierającego klasę aplikacji i na samej górze dodaj dwie linijki kodu:

```
import konfiguracja
import mysql.connector
```

Pierwsza linijka pozwoli nam korzystać ze zmiennych zapisanych w pliku „*konfiguracja*". Druga natomiast pozwoli nam korzystać z pakietu *mysql-connector-python*, który dodaliśmy do projektu w poprzedniej części instrukcji.

Zamień ciało konstruktora klasy aplikacja na następujące:

```
try:
    self.połączenie = mysql.connector.connect(
        user=konfiguracja.użytkownik,
        password=konfiguracja.hasło,
        host=konfiguracja.adres,
        database=konfiguracja.nazwaBazy
    )

except mysql.connector.Error as błąd:
    print(błąd)
```

Jak zapewne się domyślasz, funkcja `connect` nawiązuje połączenie z bazą danych. Zwróć uwagę na różnicę w działaniu pomiędzy poleceniem „*import [plik]*", a „*from [plik] import*", którą użyliśmy w pliku *main*.

W powyższym kodzie pojawiły się również dwie rzeczy, które mogą być dla Ciebie nowe: blok `try` i blok `except`. Bloki te służą do obsługi *wyjątków*. Jeżeli podczas wykonywania fragmentu kodu otoczonego blokiem `try` stanie się coś niespodziewanego (np. gdy oczekujemy, że użytkownik wpisze na klawiaturze liczbę, a zostanie podany ciąg znaków), program nie zakończy działania, tylko przekaże błąd do bloku `except`. W naszym wypadku wyjątek może wyrzucić funkcja `connect`. Z dokumentacji wiemy, że będzie on typu `Error`, dlatego też taki typ przechwytujemy w bloku `except`.

Pełna składnia bloku `try` wygląda następująco:

```
try:
    # Kod który może wyrzucić wyjątek
except TypWyjątku as nazwaZmiennejZaPomocąKtórejChcemyZłapaćWyjątek:
    # Kod obsługi wyjątku typu TypWyjątku
except:
    # Kod obsługi wyjątku dowolnego typu
else:
    # Kod wykonywany jeżeli wyjątek nie zostanie wyrzucony
finally:
    # Kod który wykona się niezależnie od tego, czy wyjątek został
    # wyrzucony, czy też nie.
```

Uruchom program. Jeżeli wszystko jest poprawnie, konsola powinna wyglądać tak samo jak poprzednio. Jeżeli jednak źle uzupełniłaś/uzupełniłeś dane dostępu do bazy, możesz

w konsoli ujrzeć treść wyjątku. Dla przykładu, jeżeli źle przepiszesz nazwę użytkownika lub hasło, konsola będzie wyglądać następująco:

```
1045 (28000): Access denied for user 'nazwaBazy'@'localhost' (using
password: YES)

Lista modułów:
======================
Brak zarejestrowanych modułów!
Wybierz moduł:
```

KLASA MODUŁU

Utwórz nowy plik i nazwij go moduł. Umieść w nim następujący kod:

```
class Moduł:
    identyfikator = 'A';
    nazwa = "Abstrakcyjny Moduł"
    aplikacja = None

    def __init__(self,identyfikator,nazwa):
        self.identyfikator = identyfikator
        self.nazwa = nazwa

    def przejmijKontrolę(self):
        print("To jest abstrakcyjna klasa modułu. Nic nie robi!")
        input("Wciśnij ENTER aby kontynuować...")
```

Kod ten powinien być dla Ciebie zrozumiały, dlatego nie ma potrzeby go szczegółowo omawiać. Klasa `Moduł` będzie klasą bazową dla bardziej zaawansowanych modułów, które będziemy tworzyć w przyszłości.

MODUŁ OBSŁUGI BAZY DANYCH

Ostatnią klasą jaką utworzymy w projekcie, będzie klasa modułu do obsługi bazy danych. Klasa ta będzie się nazywać „`ModułBazyDanych`" i dziedziczyć po klasie `Moduł`, którą opracowano w poprzednim rozdziale instrukcji. Najważniejszą zmianą w stosunku do klasy `Moduł` będzie zmieniona funkcja `przejmijKontrolę`. Będzie ona realizować następujący algorytm:

1. Wypisz na ekranie nagłówek modułu
2. Utwórz kursor czytania z bazy danych
3. Wykonuj w nieskończoność:
   3.1. Poproś użytkownika o wpisanie treści zapytania do bazy
   3.2. Jeżeli zapytanie jest równe ciągowi znaków „koniec", to:
      3.2.1. Zakończ działanie modułu i przekaż kontrolę do aplikacji
   3.3. Wyślij zapytanie do bazy danych
   3.4. Dla każdego wiersza zwróconego w odpowiedzi, wykonaj:
      3.4.1.Wypisz zawartość tego wiersza

Oczywiście należy wziąć pod uwagę obsługę wyjątków. W przeciwnym wypadku użytkownik nie będzie wiedział, dlaczego w odpowiedzi dostaje pusty wynik.

Utwórz plik „modułBazyDanych". Przepisz do niego następujący kod klasy ModułBazyDanych. Zwróć uwagę, że kod metody przejmijKontrolę implementuje powyższą listę kroków.

```
from moduł import *
import mysql.connector

class ModułBazyDanych(Moduł):
    def __init__(self):
        self.identyfikator = "BD"
        self.nazwa = "Moduł bazy danych"

    def przejmijKontrolę(self):
        print("\nModuł bazy danych:")
        print("=======================")
        zapytanie = ""
        kursor = self.aplikacja.połączenie.cursor(dictionary=True)
        while (True):
            zapytanie = input("Podaj treść zapytania SQL do bazy danych
                              (lub wpisz 'koniec'): ")
            if zapytanie.upper() == "KONIEC":
                break
            try:
                kursor.execute(zapytanie)
            except mysql.connector.Error as błąd:
                print("Błąd w zapytaniu SQL:",błąd)
            else:
                for wiersz in kursor:
                    print(wiersz)
        kursor.close()
```

Tworząc kursor czytania przekazujemy do konstruktora jeden argument – dictionary (konstruktor przyjmuje więcej argumentów, ale są one opcjonalne). Ustawienie wartości tego argumentu na „*True*" oznacza, że wyniki będą zwracane w postaci listy asocjacyjnej (czasami stosuje się również określenie „słownik"). Tablicami asocjacyjnymi zajmiemy się na kolejnych zajęciach.

Do wystosowania zapytania do bazy danych służy funkcja execute. Jako argument przekazujemy treść zapytania, które chcemy wykonać.

Wynik zapytania jest przechowywany w zmiennej *kursor*. Możemy przeglądnąć zwrócone wiersze w taki sam sposób, w jaki przeglądamy elementy w liście. Kończąc pracę z kursorem dobrze jest wywołać funkcję close.

Żeby z modułu można było skorzystać, musimy jeszcze go zarejestrować w pliku *main*. W tym celu, na samej górze pliku należy dopisać komendę *import*, a następnie wywołać metodę zarejestrujModuł klasy Aplikacja, podając odpowiednią wartość argumentu. Cały kod zawarty w pliku *main* powinien wyglądać w następujący sposób:

```
from aplikacja import *
from modułBazyDanych import *

# utwórz aplikację
aplikacja = Aplikacja()

# zarejestruj moduły
aplikacja.zarejestrujModuł(ModułBazyDanych())
```

```
# dopóki aplikacja działa
while(aplikacja.działa):
    # wyświetl menu
    aplikacja.wyświetlMenu()

    # pozwól użytkownikowi wybrać moduł i przekaż kontrolę
    wybór = input("Wybierz moduł: ")
    aplikacja.wybierzModuł(wybór)
```

ZADANIA DO SAMODZIELNEGO WYKONANIA

ZADANIE 1

Uruchom aplikację i zadaj zapytanie o treści „`SELECT * FROM users`". Przeanalizuj wynik. W szczególności zwróć uwagę w jakiej formie są wyświetlane wyniki i jakie kolumny zawiera pojedynczy wiersz. Następnie zadaj zapytanie o treści „`SELECT * FROM devices`" i przeanalizuj różnice.

ZADANIE 2 Ⓟ

Dodaj nowy moduł do menu (podpowiedź: nie musisz tworzyć dodatkowej klasy). Ustaw jego identyfikator na „KONIEC" i nazwę na „Zakończ program". Przetestuj działanie programu.

ZADANIE 3 (PLUS MOŻE ZDOBYĆ KAŻDA OSOBA, KTÓRA SKOŃCZY ZADANIE)

Utwórz w osobnym pliku nowy moduł aplikacji. Ustaw jego identyfikator na „CONN" i nazwę na „Konfiguracja połączenia". Zaimplementuj funkcję `przejmijKontrolę` w taki sposób, aby pytała użytkownika o dane połączenia (nazwę użytkownika, hasło i adres bazy), a następnie zmieniała wartości odpowiednich zmiennych w pliku „konfiguracja".

ZADANIE 4 Ⓟ (NAGRADZANE 2 PLUSAMI)

Utwórz w osobnym pliku nowy moduł aplikacji. Ustaw jego identyfikator na „INFO" i nazwę na „Informacje o połączeniu". Zaimplementuj funkcję `przejmijKontrolę` w taki sposób, aby wypisała na ekranie nazwę użytkownika, hasło, nazwę bazy danych oraz stan połączenia w postaci „*status połączenia: DZIAŁA*" lub „*status połączenia: BRAK/NIEUDANE*". Zadbaj o to, by wypisany tekst był estetyczny i łatwy do zrozumienia dla użytkownika. Po wypisaniu informacji, przekaż kontrolę z powrotem do aplikacji. Zarejestruj moduł w aplikacji.

Podpowiedź: możesz sprawdzić stan połączenia przyrównując wartość pola *połączenie* do słowa kluczowego `None`:

```
if self.aplikacja.połączenie == None:
    print("Nie udało się połączyć z bazą danych!")
```