Nauka programowania w językach Scratch i Python

Python – poziom średniozaawansowany

Materiał dla nauczycieli

[1. Podstawy języka Python 1](#_Toc1174804807)

[1.1. Język Python, jego rola i podstawowe narzędzia 2](#_Toc1648901842)

[1.2. Przygotowanie środowiska w różnych systemach operacyjnych 9](#_Toc99818194)

[1.3. Uruchamianie kodu w konsoli 24](#_Toc1009348488)

[1.4. Typy podstawowe oraz metody wczytywania danych 31](#_Toc1763172129)

[1.5. Instalacja IDE oraz omówienie menedżera PIP 39](#_Toc253195940)

[1.6. Formatowanie kodu oraz korzystanie z IDE 48](#_Toc2136505073)

[1.7. Deklarowanie zmiennych i rola komentarzy 50](#_Toc372634602)

[1.8. Logika warunkowa 53](#_Toc1593747456)

[1.9. Logika warunkowa w praktyce 55](#_Toc407412659)

[1.10. Konwersje liczb 56](#_Toc1584034702)

[1.11. Działania matematyczne 57](#_Toc1599063591)

[1.12. Podsumowanie 58](#_Toc600073837)

[1.13. Zadania podsumowujące moduł 59](#_Toc1274929051)

[1.14. Dodatek 61](#_Toc1478437518)

[2. Struktury danych 70](#_Toc1315283998)

[2.1. Zapoznanie z obiektami: Listy, krotki i słowniki 71](#_Toc653967934)

[2.2. Tworzenie napisów i działanie na łańcuchach znaków 72](#_Toc1803793555)

[2.3. Podsumowanie 73](#_Toc1163400758)

[2.4. Zadania podsumowujące moduł 73](#_Toc1408934280)

[2.5. Dodatek 75](#_Toc689924129)

[3. Pętle i iteracje 78](#_Toc1116069810)

[3.1. Zrozumienie i zastosowanie pętli 79](#_Toc2039064480)

[3.2. Rekurencja i zasięg zmiennych 82](#_Toc1287060362)

[3.3. Iteracja przez elementy w strukturach danych 84](#_Toc1757911980)

[3.4. Podsumowanie 86](#_Toc1122615486)

[3.5. Zadania podsumowujące moduł 86](#_Toc252880168)

[3.6. Dodatek 88](#_Toc61295196)

[4. Funkcje i moduły 92](#_Toc1559077101)

[4.1. Zastosowanie funkcji 92](#_Toc1983558216)

[4.2. Popularne moduły (biblioteki) 94](#_Toc334854476)

[4.3. Tworzenie własnych modułów 95](#_Toc1465362476)

[4.4. Podsumowanie 96](#_Toc981796600)

[4.5. Zadania podsumowujące moduł 97](#_Toc1333358921)

[4.6. Dodatek 99](#_Toc1625232975)

[5. Klasy, obsługa plików, operacje wejścia/wyjścia, integracja z zewnętrznymi urządzeniami 102](#_Toc1085568097)

[5.1. Wprowadzenie do programowania obiektowego i klas 103](#_Toc1468122173)

[5.2. Obsługa popularnych błędów i własnych klas wyjątków 104](#_Toc672637072)

[5.3. Obsługa plików – odczyt i zapis danych 107](#_Toc1186191912)

[5.4. Przetwarzanie danych 108](#_Toc1828986394)

[5.5. Integracja z zewnętrznymi urządzeniami 110](#_Toc1109439150)

[5.6. Podsumowanie 111](#_Toc361674896)

[5.7. Zadania podsumowujące moduł 112](#_Toc84868855)

[5.8. Dodatek 114](#_Toc150941672)

[6. Tworzenie graficznych interfejsów użytkownika 121](#_Toc44993156)

[6.1. Wprowadzenie do Tkinter 122](#_Toc2002484524)

[6.2. Tworzenie bardziej zaawansowanego interfejsu – formularz i pola wejściowe 123](#_Toc221863142)

[6.3. Wizualizacja danych – prezentacja tabel i wykresów 125](#_Toc1934988796)

[6.4. Podsumowanie 126](#_Toc1650820676)

[6.5. Zadania podsumowujące moduł 126](#_Toc1205559126)

[6.6. Dodatek 128](#_Toc1022479928)

Spis treści

# Podstawy języka Python

E-skrypt stanowi kompleksowy materiał dydaktyczny, obejmujący podstawową teorię programowania w języku Python, zilustrowaną przykładami praktycznymi oraz zadaniami do samodzielnego rozwiązania. W każdym rozdziale czytelnik znajdzie szczegółowe omówienie tematów, wzbogacone o diagramy, fragmenty kodu i przykłady wyników działania listingów w formie zrzutów ekranu. Zadania o zróżnicowanym poziomie trudności, w tym bardziej wymagające (oznaczone znakiem \*), umożliwiają utrwalenie wiedzy i rozwój umiejętności w rozwiązywaniu problemów programistycznych, przygotowując do pracy z językiem Python.

## Język Python, jego rola i podstawowe narzędzia

### Wprowadzenie

Pierwszy moduł wprowadza uczestników w podstawy języka Python, koncentrując się na pracy w konsoli systemowej, co pozwala skupić się na podstawowych mechanizmach języka bez konieczności korzystania z zaawansowanych środowisk programistycznych. Uczestnicy poznają zasady instalacji środowiska Python na różnych systemach operacyjnych, tworzenia skryptów, a także pracy z interaktywną konsolą. Moduł obejmuje zagadnienia teoretyczne, takie jak formatowanie, struktura programu i typy danych, jak i przykłady oraz praktyczne ćwiczenia do samodzielnego zrealizowania. Rozdział zawiera również operacje matematyczne i manipulację tekstem. Dodatkowo uwzględniono korzystanie z bibliotek języka Python oraz zarządzanie nimi za pomocą menedżera PIP.

#### Struktura kursu i sposób nauki

E-skrypt składa się z **6 modułów**, z których każdy stanowi rozszerzenie wiedzy na temat języka Python, wprowadzając coraz bardziej zaawansowane zagadnienia. Moduły podzielone są na tematyczne lekcje, które systematycznie rozwijają umiejętności uczestników. Każda lekcja zawiera wprowadzenie, część teoretyczną dotyczącą programowania, zestaw praktycznych zadań, które pomagają utrwalić zdobytą wiedzę oraz podsumowania.

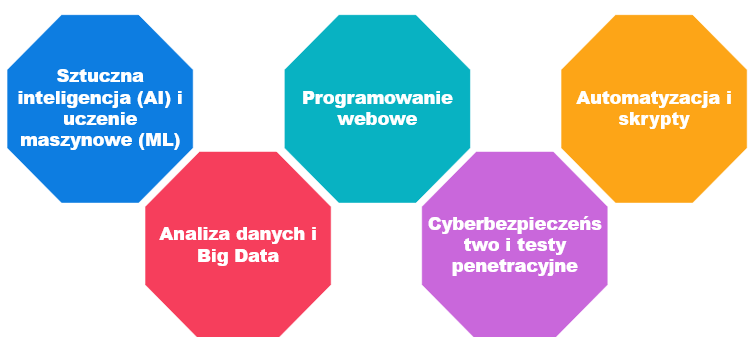
Sposób nauki opiera się na realizacji przykładów oraz samodzielnym wykonywaniu zadań utrwalających, co pozwala na aktywne przyswajanie materiału. Dla osób bardziej zaawansowanych, w każdej lekcji przygotowano dodatkowe zadania (oznaczone \*) o wyższym stopniu trudności, które umożliwiają pogłębienie wiedzy i rozwój umiejętności programistycznych.

#### Zastosowanie języka

Język Python to jeden z najpopularniejszych języków programowania, ceniony za swoją prostotę, czytelność oraz wszechstronność. Dzięki intuicyjnej składni i bogatemu ekosystemowi bibliotek, jest często wybierany zarówno przez początkujących programistów, jak i doświadczonych specjalistów.

Jego zastosowania obejmują szeroki wachlarz dziedzin – od analizy danych i sztucznej inteligencji, przez automatyzację procesów, aż po tworzenie zaawansowanych aplikacji webowych. Popularność języka Python wynika również z jego dynamicznie rozwijającej się społeczności, która nieustannie tworzy nowe narzędzia i udoskonala istniejące rozwiązania.

Jest także doskonałym językiem do nauki programowania dla dzieci i młodzieży, ponieważ jego składnia jest przejrzysta, a dostępne materiały edukacyjne pozwalają na szybkie przyswojenie podstaw i rozwijanie umiejętności programistycznych w przystępny sposób.



Zastosowanie języka python w różnych dziedzinach

#### Wady i zalety języka Python

Python jest językiem przyjaznym dla początkujących, wyróżniającym się czytelną składnią i bogatym ekosystemem bibliotek, które ułatwiają naukę programowania. W porównaniu do języków takich jak C++ czy Java jego przejrzysta struktura sprawia, że podstawowe koncepcje programistyczne są łatwiejsze do zrozumienia. Możliwość natychmiastowego testowania kodu w trybie interaktywnym oraz szybkie osiąganie widocznych rezultatów zwiększają motywację do nauki i sprawiają, że Python jest doskonałym wyborem jako pierwszy język do nauczania w szkołach.

Jednak interpretowany charakter języka sprawia, że działa wolniej niż języki kompilowane, co może stanowić ograniczenie w przypadku aplikacji wymagających wysokiej wydajności. Dodatkowo dynamiczne typowanie, choć ułatwia pisanie kodu, może prowadzić do trudniejszych do wykrycia błędów w większych projektach.

#### Wykorzystanie języka Python – interaktywnie czy przez skrypt?

Python oferuje dwa główne sposoby uruchamiania kodu: tryb interaktywny oraz uruchamianie skryptów zapisanych w plikach *.py*. Każdy z tych sposobów ma swoje zalety i ograniczenia, a ich wybór zależy od kontekstu i celu pracy z językiem.

Tryb interaktywny pozwala na szybkie testowanie kodu bez tworzenia plików. W Debian/Linux uruchamia się go przez komendę *python3* w terminalu, a w systemie Windows przez *python* lub *python3* w konsoli lub PowerShell.

Zalety oraz ograniczenia trybu interaktywnego.

|  |  |
| --- | --- |
| **Zalety trybu interaktywnego** | **Ograniczenia trybu interaktywnego** |
| Szybkie testowanie kodu | Brak trwałości kodu |
| Łatwość eksperymentowania | Niepraktyczny dla dużych projektów |
| Brak potrzeby zapisywania plików | Ograniczone wsparcie dla organizacji kodu |

Drugi sposób to pisanie kodu w plikach *.py*, które następnie można uruchomić poprzez interpreter. Do edycji tych plików można używać prostych edytorów, takich jak Notatnik, jak również rozbudowanych środowisk programistycznych, takich jak Visual Studio Code czy Thonny, które oferują dodatkowe funkcje ułatwiające pracę z kodem.

Zalety oraz ograniczenia plików .py

|  |  |
| --- | --- |
| **Zalety skryptów Python** | **Ograniczenia skryptów Python** |
| Trwałość i możliwość ponownego użycia | Wolniejszy cykl testowania |
| Łatwość organizacji kodu | Początkowa konfiguracja |
| Lepsza integracja z innymi narzędziami |  |

Najlepszym podejściem jest korzystanie z obu metod w zależności od aktualnych potrzeb – tryb interaktywny do eksploracji i testów, a skrypty do budowania kompletnych aplikacji.

### Algorytmika – czym jest?

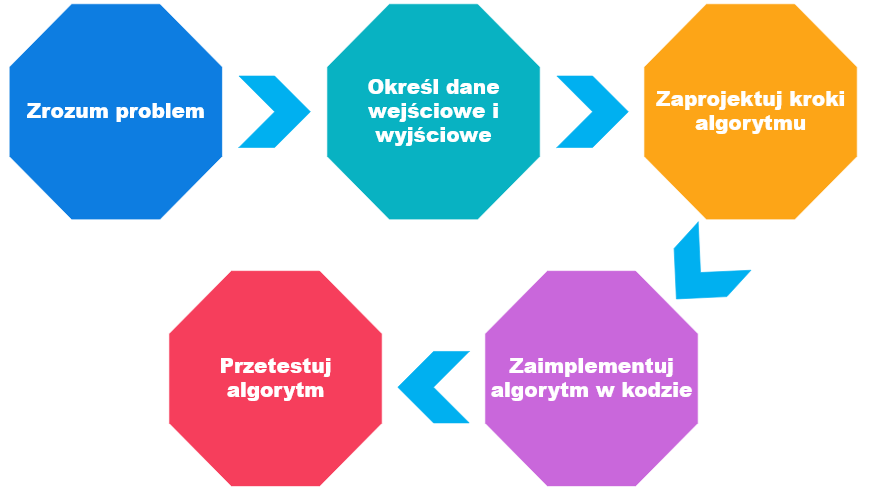
Algorytm to uporządkowany zbiór kroków lub instrukcji prowadzących do rozwiązania określonego problemu. Może być zapisany w postaci schematu blokowego, opisu słownego lub kodu komputerowego i powinien być jednoznaczny oraz efektywny. W języku Python algorytmy można implementować przy użyciu struktur danych i instrukcji sterujących, np. sortowanie listy liczb można wykonać za pomocą algorytmu sortowania bąbelkowego lub wbudowanej funkcji.

Aby utworzyć algorytm musisz wiedzieć, co chcesz osiągnąć. Przykładowo, jeśli chcesz znaleźć największą liczbę w liście, musisz wiedzieć, jakie są dane wejściowe i jaki powinien być wynik. Załóżmy, że poszukujemy maksymalnej liczby z zbioru, gdzie:

* Przykładowe wejście: Lista liczb [3, 7, 1, 9, 5].
* Przykładowe wyjście: Największa liczba, czyli 9.

Kolejne kroki można przedstawić jako:

1. Weź pierwszą liczbę jako największą.
2. Przejdź przez resztę listy i porównuj każdą liczbę z największą.
3. Jeśli znajdziesz większą, zamień największą na tę liczbę.
4. Na końcu zwróć największą wartość.



Schemat procesu projektowania algorytmu od analizy problemu po testowanie rozwiązania

Warto znać pojęcia takie jak złożoność obliczeniowa i pamięciowa, które są nieodłącznymi parametrami algorytmów. Są to miary określające, jak algorytm zachowuje się pod względem czasu wykonania oraz zużycia pamięci w zależności od wielkości danych wejściowych.

W trakcie kursu uczestnicy poznają różne algorytmy – od najprostszych po bardziej zaawansowane metody rozwiązywania problemów. Choć algorytmy są niezależne od konkretnego języka programowania, w ramach zajęć będą one realizowane w języku Python.

### Podstawowe narzędzia kursu

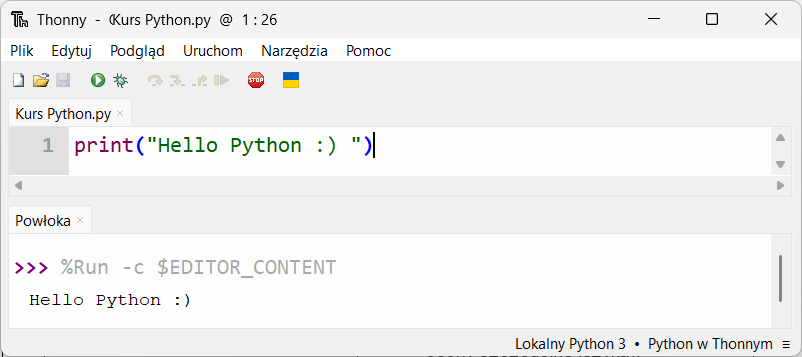
W trakcie kursu będziemy korzystać z różnych narzędzi wspomagających naukę i pracę z językiem Python. Obejmują one zarówno systemy operacyjne, na których działa Python, jak i środowiska programistyczne oraz edytory, które ułatwiają pisanie i uruchamianie kodu. W początkowych etapach nauki szczególnie skupimy się na pracy w konsoli systemowej (terminal w Linux, CMD/PowerShell w Windows), co pozwoli zrozumieć mechanizmy interpretera języka Python. Dodatkowo, poznamy różnice między trybem interaktywnym a skryptowym.

#### Systemy operacyjne (Windows oraz Ubuntu)

Python jest językiem wieloplatformowym, co oznacza, że może działać na różnych systemach operacyjnych, takich jak Windows, Linux i macOS. W trakcie kursu uczestnicy będą głównie pracować w środowisku Windows, ale poznają również sposoby instalacji i uruchamiania interpretera Python w systemie Linux (Ubuntu), korzystając z maszyny wirtualnej. Umiejętność pracy w różnych środowiskach pozwoli uczestnikom na elastyczne dostosowanie się do różnych warunków pracy i zapewni lepsze zrozumienie narzędzi wykorzystywanych w programowaniu.

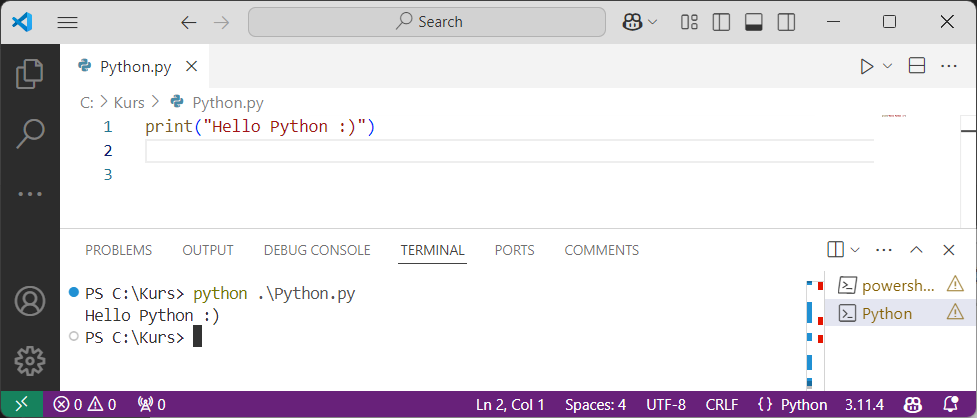
#### Środowiska i edytory (Thonny, Visual Studio Code)

W trakcie kursu uczestnicy będą korzystać z różnych edytorów, środowisk programistycznych i narzędzi ułatwiających pracę z językiem Python. Każde z nich ma swoje zalety i znajduje zastosowanie w różnych aspektach programowania. Thonny – prosty edytor stworzony z myślą o nauce języka Python, szczególnie polecany dla początkujących. Thonny jest udostępniany na licencji MIT, co oznacza, że jest darmowy, open-source i można go dowolnie modyfikować oraz wykorzystywać zarówno do celów edukacyjnych, jak i komercyjnych.



Thonny – prosty i darmowy edytor stworzony z myślą o nauce języka Python

Znaczenie bardziej rozbudowanym edytorem jest Visual Studio Code (VS Code) – lekki, wszechstronny edytor z obsługą rozszerzeń, który zapewnia podświetlanie składni, autouzupełnianie kodu i integrację z systemem kontroli wersji. VS Code jest rozwijany przez firmę Microsoft i udostępniany na licencji open-source MIT, co pozwala na jego darmowe używanie oraz modyfikowanie przez społeczność.

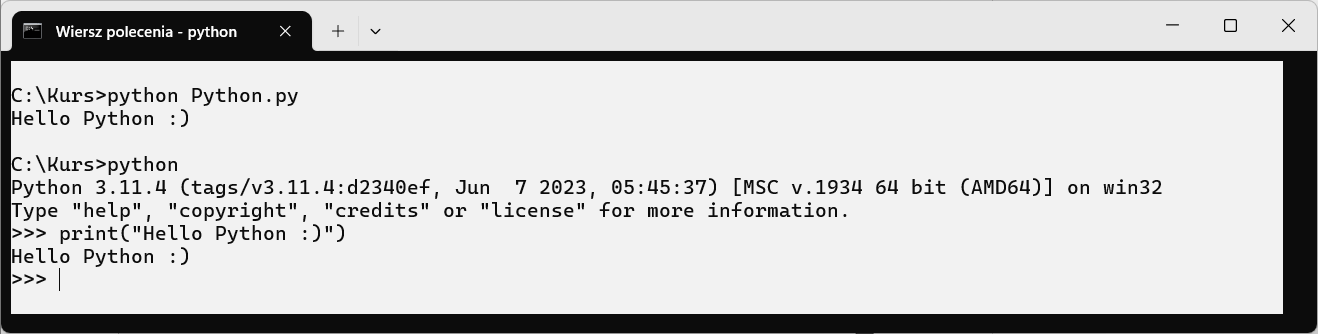


Visual Studio Code (VS Code) – lekki edytor z obsługą rozszerzeń

Do popularnych narzędzi powszechnie stosowanych do tworzenia aplikacji w języku Python należą m.in Jupyter Notebook, który jest interaktywnym środowiskiem idealnym do analizy danych i uczenia maszynowego oraz PyCharm - zaawansowane środowisko IDE oferujące rozbudowane funkcje ułatwiające programowanie, debugowanie i zarządzanie projektami.

#### Uruchamianie kodu języka Python w terminalu

Terminal jest podstawowym narzędziem do uruchamiania skryptów napisanych w języku Python. Wystarczy wpisać komendę *python nazwa\_pliku.py*, aby wykonać kod zapisany w pliku. Oprócz tego Python oferuje tryb interaktywny, który pozwala na wpisywanie poleceń i natychmiastowe otrzymywanie wyników, co jest szczególnie przydatne do testowania krótkich fragmentów kodu i eksperymentowania z językiem.



Uruchamianie poleceń języka Python w konsoli (CMD systemu Windows

|  |  |
| --- | --- |
|  | Skróty klawiszowe uruchamiające terminal w systemie Windows:   * Skrót Win + R, następnie wpisz cmd, następnie Enter * Skrót Win + X, następnie A – PowerShell |

|  |  |
| --- | --- |
|  | Skróty klawiszowe uruchamiające terminal w systemie Ubuntu:  Skrót Ctrl + Alt + T dla Ubuntu |

### Zadania

**Zadanie 1.1.1. Konsola systemu Windows**

* Uruchom konsolę systemu z prawami administratora (Jeśli cmd został uruchomiony z uprawnieniami administratora, w tytule okna zazwyczaj pojawia się przedrostek „Administrator”).

**Zadanie 1.1.2. Visual Studio Code**

* Uruchom Visual Studio Code oraz zmień motyw na jasny.

**Zadanie 1.1.3. Kolor konsoli systemowej**

* W konsoli cmd zmień kolor tła (wpisz komendę *color F0*).

**Zadanie 1.1.4. \* Terminal w systemie Linux**

Uruchom terminal w systemie Windows, następnie komendą dir sprawdź zawartość katalogu.

**Zadanie 1.1.5. \* Uruchamianie cmd skryptem**

* Uruchom nową konsolę z białym tłem (*color F0*) za pomocą języka Python.
* Wywołana konsola ma wypisać wersje języka Python.

\*Wymaga zastosowania subprocess.

## Przygotowanie środowiska w różnych systemach operacyjnych

### Instalacja środowiska Python na systemie Linux

Przykład instalacji środowiska Python na systemie Linux wymaga posiadania tego systemu zainstalowanego natywnie na komputerze lub korzystania ze środowiska wirtualnego emulującego Linux. W poniższym przykładzie zostanie użyte oprogramowanie Oracle VirtualBox, które zostało już skonfigurowane i zawiera odpowiedni obraz systemu.

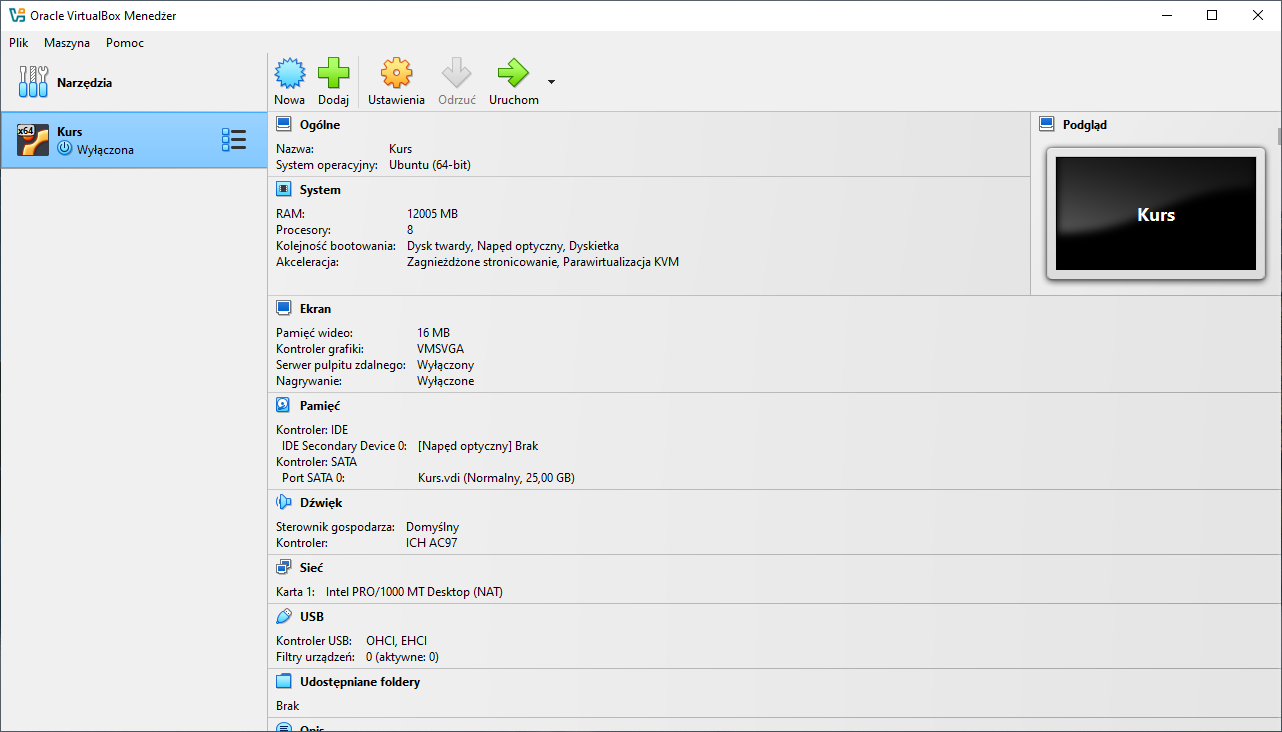
Aby skorzystać z systemu, należy uruchomić aplikację z poziomu pulpitu lub wpisać jej nazwę w pasku wyszukiwania systemu.

|  |  |
| --- | --- |
|  |  |

Ikona emulatora systemu Oracle VirtualBox

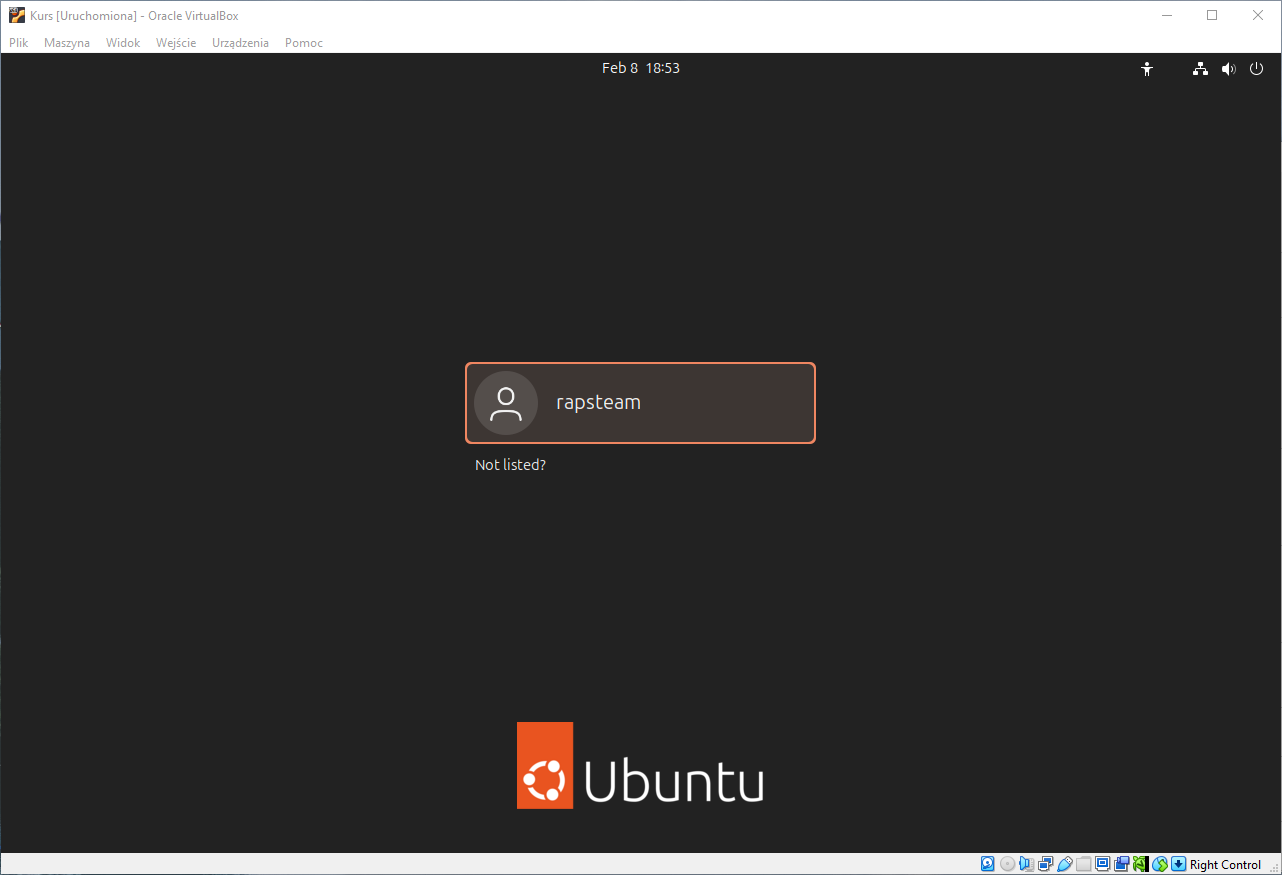
Po uruchomieniu, pojawi się okno z widokiem dostępnych i skonfigurowanych maszyn wirtualnych, gotowych do uruchomienia. Aby wystartować jedną z nich, należy wybrać pozycję z listy po prawej stronie widoku o nazwie *Kurs* i nacisnąć klawisz *Uruchom* widoczny w górnej części okna.

W centralnej części ekranu widoczne są dane konfiguracyjne oraz przypisane zasoby. W razie potrzeby można je zmodyfikować w menu *Ustawienia*, znajdującym się nad tymi informacjami.



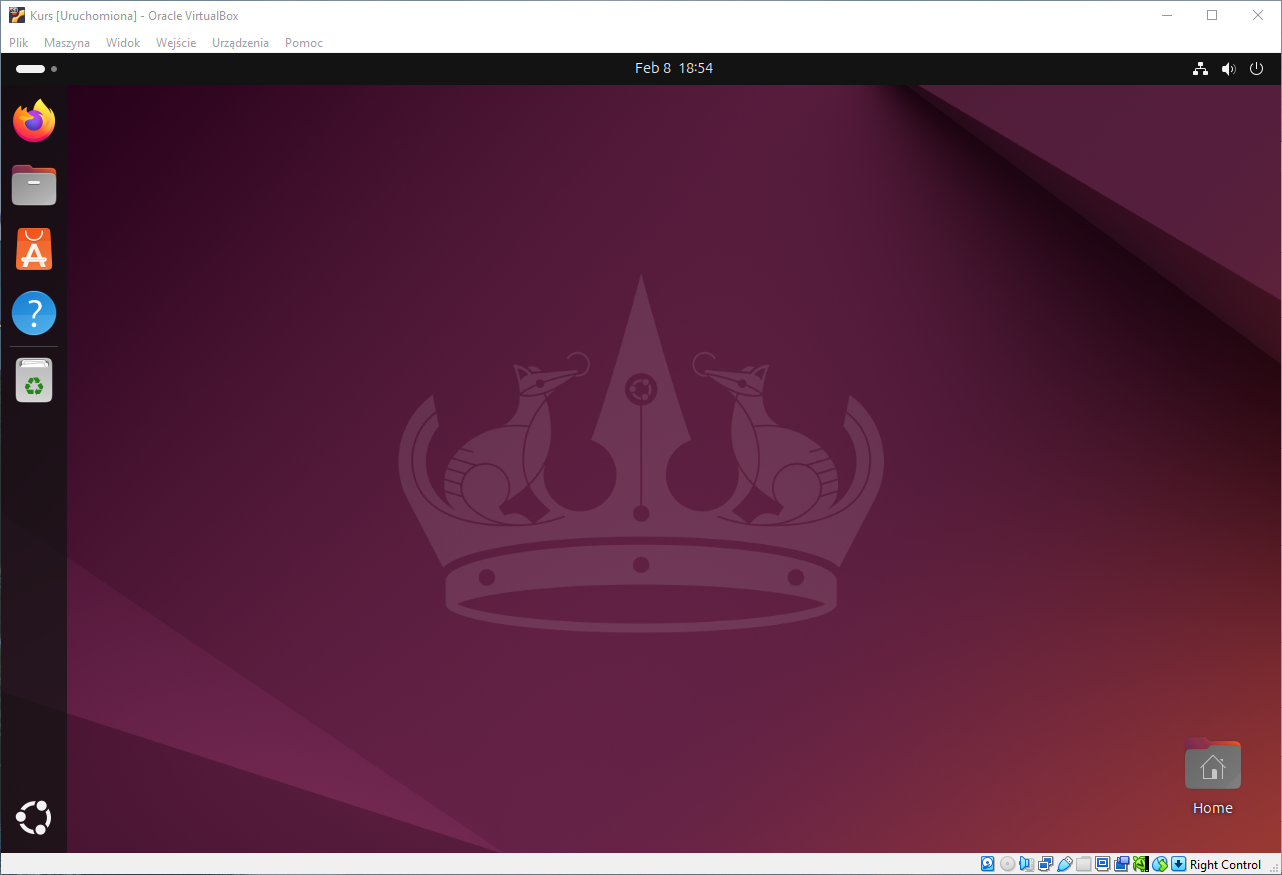
Okno wyboru skonfigurowanych maszyn wirtualnych

Po wciśnięciu klawisza *Uruchom* pojawi się dodatkowe okno, emulujący monitor uruchamianego systemu. Po chwili, pokaże się okno logowania użytkownika, w którym należy podać hasło logowania podane przez trenera.



Okno logowania użytkownika na systemie Ubuntu

Po pomyślnym logowaniu, pojawi się pulpit gotowy do pracy. System uruchomi środowisko graficzne, udostępniając menu, ikony oraz pasek zadań, które umożliwiają szybki dostęp do najważniejszych aplikacji. Użytkownik może teraz rozpocząć pracę, personalizując ustawienia pulpitu oraz korzystając z dostępnych narzędzi systemowych.

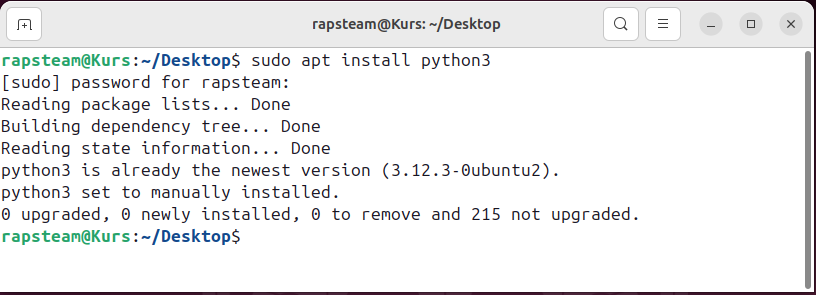


Pulpit systemu Ubuntu

Aby zainstalować środowisko Python w systemie Linux, należy otworzyć terminal za pomocą skrótu klawiszowego *CTRL ALT + T* lub z poziomu pulpitu naciskając prawy przycisk myszy i wybierając *Open new terminal*. Następnie należy wpisać poniższe polecenie.

Instalacja interpretera Python 3 w systemie Ubuntu

|  |
| --- |
| sudo apt install python3 |



Instalacja pakietu Python w systemie Ubuntu

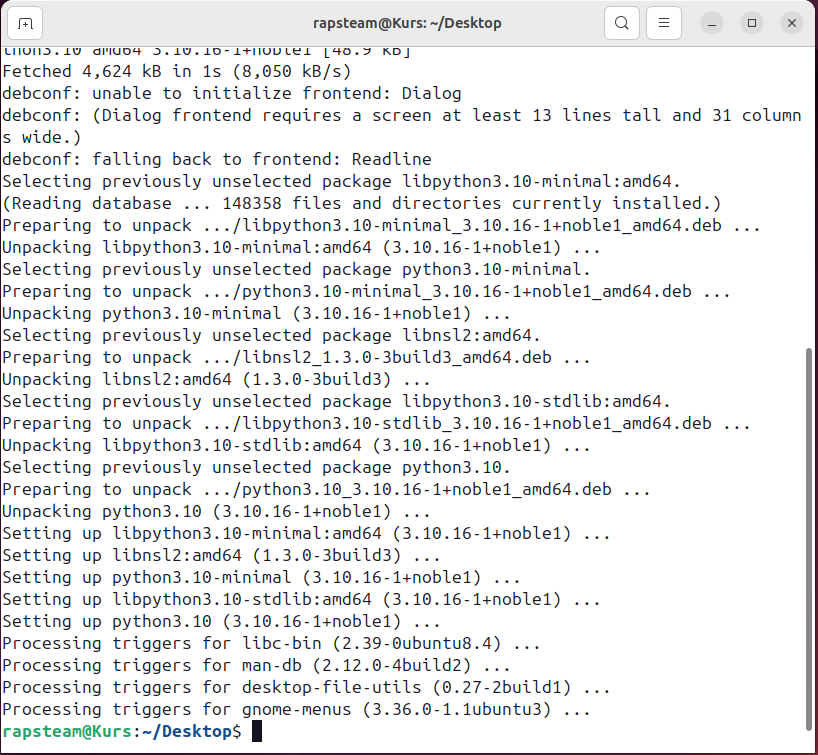
Możliwe jest także wskazanie konkretnej wersji pakietu Python do instalacji. Dzięki instalacji wielu wersji pakietu, można zapewnić kompatybilność różnych projektów, które wymagają specyficznych wersji interpretera i bibliotek. Pozwala to uniknąć konfliktów między zależnościami oraz zapewnia stabilność i przewidywalność działania aplikacji.

Instalacja pakietu Python w wersji 3.10 w systemie Ubuntu

|  |
| --- |
| sudo apt install python3.10 |

Po wydaniu komendy system pobierze i zainstaluje interpreter języka Python w wersji 3.10 wraz z wszystkimi niezbędnymi zależnościami. Jeśli dany pakiet wymaga dodatkowych bibliotek systemowych, systemowy menedżer pakietów automatycznie je doinstaluje.

|  |  |
| --- | --- |
|  | Zwróć uwagę na użycie *sudo*, które tymczasowo podnosi uprawnienia administratora (po podaniu hasła) w systemach opartych na dystrybucji Debian (np. Ubuntu), umożliwiając instalację oprogramowania przez *apt*. |

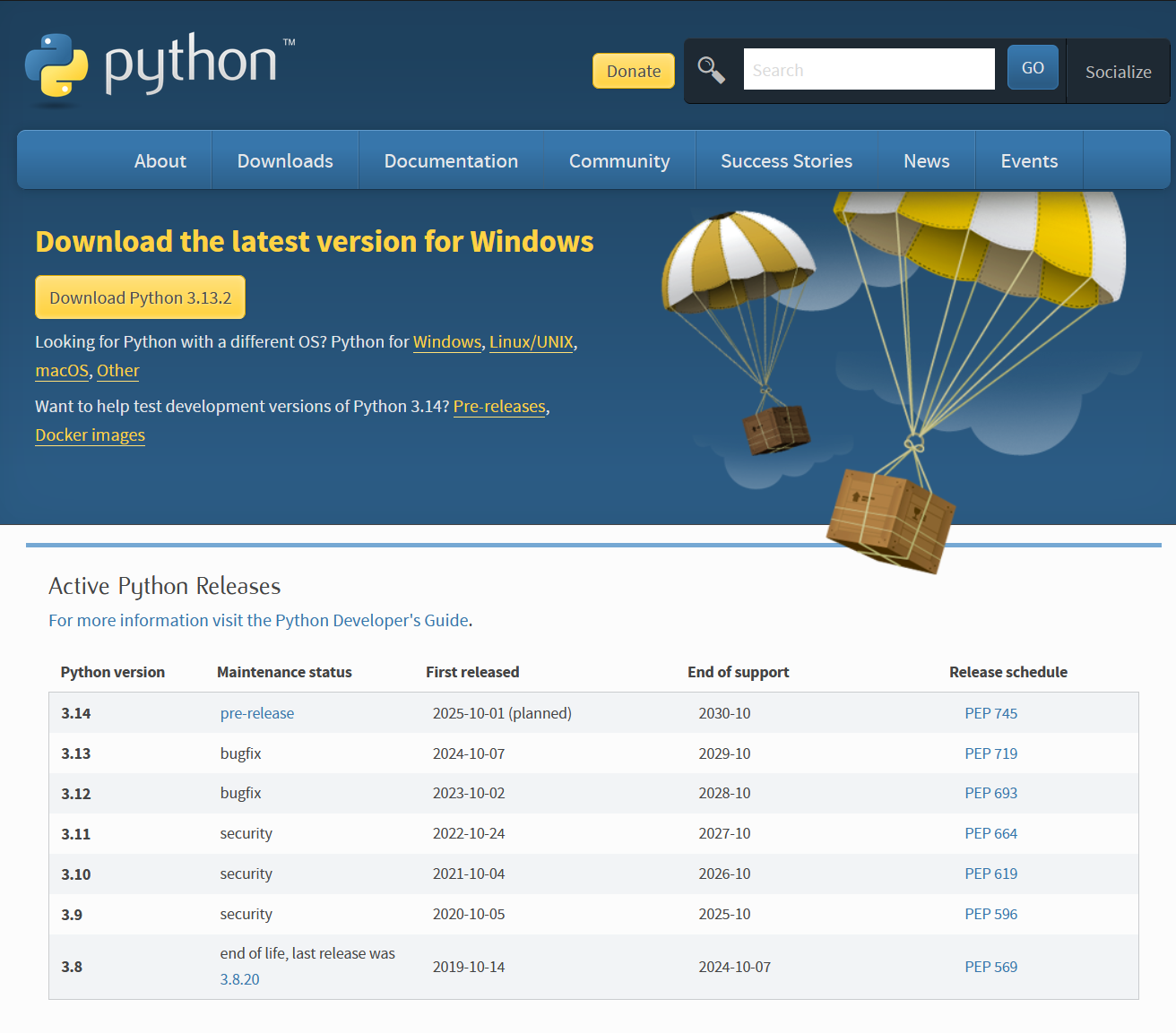


Proces instalacji pakietu Python w systemie Ubuntu

W ten sposób przygotowaliśmy środowisko z poprawnie zainstalowanym interpreterem Python w systemie Ubuntu. Umożliwia ono tworzenie i uruchamianie skryptów oraz kodu, a także instalację dodatkowych bibliotek, rozwój aplikacji i ich testowanie.

### Instalacja środowiska Python na systemie Windows

Aby zainstalować środowisko Python, należy najpierw pobrać je ze strony projektu[[1]](#footnote-2). W tym celu należy przejść do zakładki *Downloads*, gdzie dostępne są różne wersje interpretera. Na stronie znajdziemy najnowszą, rekomendowaną wersję instalatora, a także archiwalne wersje, sięgające aż Python 2.0 z 2001 roku.



Oficjalna strona projektu Python

|  |  |
| --- | --- |
|  | Na powyższym ekranie aktualna wersja to 3.13.2. Ciągły rozwój języka sprawia, że regularnie pojawiają się nowe wersje, dlatego rekomendowana wersja może ulec zmianie. |

Po pobraniu pliku instalatora z rozszerzeniem *.exe* i uruchomieniu go, pokaże się ekran instalacji. Do dyspozycji są dwie opcje instalacji:

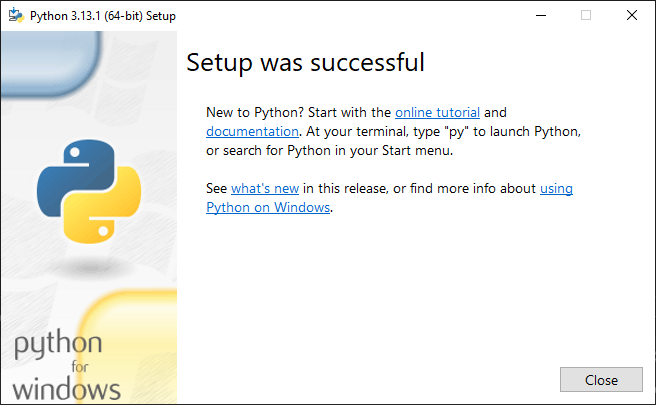
* Instalacja uproszczona (*Install Now*) – instaluje interpreter w domyślnej, wskazanej przez program lokalizacji systemowej oraz załącza do tego procesu szereg podstawowych, najpopularniejszych bibliotek dla tego języka, menedżer pakietów PIP oraz ustawia skojarzenie plików z rozszerzeniem *.py* jako pliki uruchomieniowe przez interpreter języka Python.
* Instalacja spersonalizowana (*Customize installation*) – instalacja przeznaczona dla bardziej zaawansowanych użytkowników.



Okno instalatora interpretera Python (win)

Dodatkowo, należy zaznaczyć pole wyboru *Add python.exe to PATH* na dole okna, dzięki któremu nie będzie trzeba ręcznie dodawać ścieżki, na której zostanie zainstalowany interpreter, aby ten był widoczny w terminalu lub środowisku programistycznym.

Po kliknięciu *Install Now* pojawi się okno postępu instalacji. Po jej zakończeniu wyświetli się komunikat informujący o pomyślnym ukończeniu procesu.



Ekran końcowy instalatora Python potwierdzający zakończenie instalacji pomyślnym komunikatem.

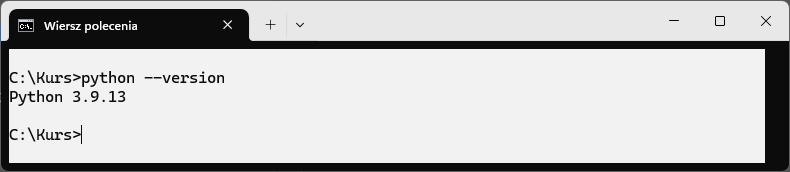
### Weryfikacja wersji interpretera języka Python

Gdy proces instalacji zakończy się pomyślnie, zarówno w systemie Windows, jak i Linux, można zweryfikować, czy wywołanie polecenia *python* uruchamia interpreter języka. W tym celu należy skorzystać z terminala systemowego.

|  |  |
| --- | --- |
|  | W przypadku systemu Windows, jeśli Python został zainstalowany po raz pierwszy, może być konieczne ponowne uruchomienie komputera przed pierwszym uruchomieniem interpretera. |

Wyświetlenie obecnie zainstalowanej wersji interpretera Python

|  |
| --- |
| python --version |



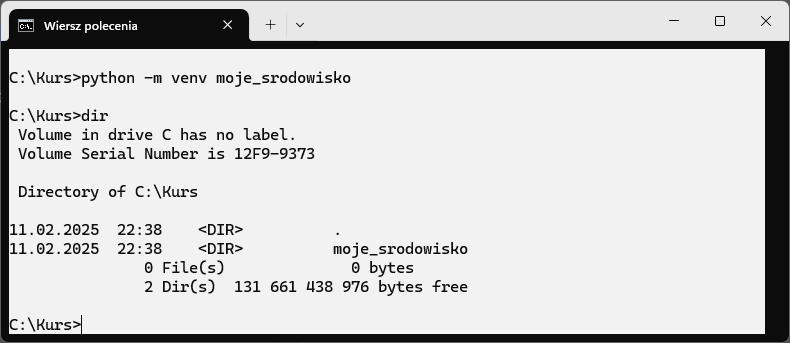
Weryfikacja instalacji Python w systemie Windows

### Tworzenie wirtualnych środowisk

Wirtualne środowiska Python są często używane do izolowania zależności różnych aplikacji. Do utworzenia wirtualnego środowiska wykorzystamy moduł *venv*. W tym celu należy otworzyć terminal systemowy i przejść do katalogu, w którym środowisko ma zostać utworzone. Następnie wystarczy wykonać polecenie stosując *venv*, aby zainicjować proces jego tworzenia.

Komenda wywołująca stworzenie środowiska wirtualnego dla języka Python

|  |
| --- |
| python -m venv moje\_srodowisko |

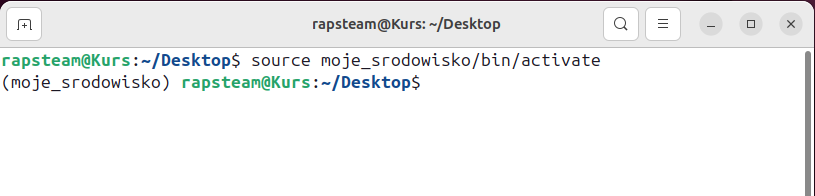


Tworzenie wirtualnego środowiska za pomocą pakietu venv

W wyniku wykonania tej komendy powstanie katalog zawierający wszystkie niezbędne pliki, które umożliwiają izolację zależności i konfiguracji projektu od globalnej instalacji.

Polecenie aktywujące środowisko wirtualne w systemie Ubuntu

|  |
| --- |
| source moje\_srodowisko/bin/activate |

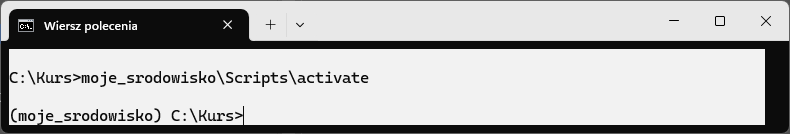


Aktywacja środowiska wirtualnego w systemie Ubuntu

Po utworzeniu środowiska należy je aktywować. W dystrybucji Debian aktywację wykonuje się poleceniem *source,* natomiast w systemie Windows uruchamiając skrypt *activate*.

Polecenie aktywujące środowisko wirtualne w systemie Windows

|  |
| --- |
| moje\_srodowisko\Scripts\activate |



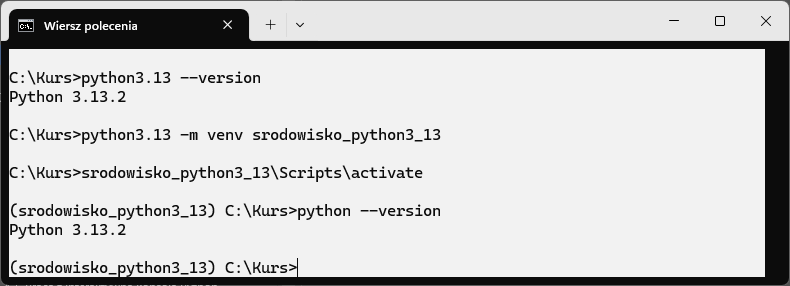
Aktywacja środowiska wirtualnego w systemie Windows

Zmiana wyglądu tzw. promptu terminala potwierdza, że środowisko zostało aktywowane i jest gotowe do pracy. Aby je dezaktywować, wystarczy wykorzystać polecenie *deactivate*.

### Zarządzanie wersjami

Python jest językiem dynamicznym, który nieustannie się rozwija, a wraz z nim zmieniają się także biblioteki i zależności wykorzystywane w projektach. Każda aplikacja może wymagać innej wersji zarówno samego języka Python, jak i jego pakietów, co może prowadzić do konfliktów.

Wyobraźmy sobie sytuację, w której pracujemy nad dwoma projektami: jeden wymaga starszej wersji popularnej biblioteki, a drugi jej najnowszego wydania. Jeśli zainstalujemy te wersje globalnie, może dojść do kolizji, co skutkuje błędami i problemami z działaniem aplikacji. Rozwiązaniem tego problemu jest tworzenie izolowanych środowisk wirtualnych, które pozwalają każdemu projektowi działać w jego własnym, odrębnym ekosystemie.



Przykład utworzenia środowiska wirtualnego dla wersji 3.13 języka Python

Dzięki temu, że można zarządzać wersjami języka i izolować środowiska możemy uniknąć nieprzewidzianych błędów, zapewnić stabilność kodu i łatwiej kontrolować wersje używanych narzędzi. W praktyce oznacza to, że nasz kod działa dokładnie tak samo na różnych komputerach, niezależnie od zmian w systemie czy aktualizacji pakietów.

### Struktura plików .py

Python jest językiem, który wymaga konsekwentnego formatowania kodu, ponieważ używa wcięć do określania struktury programu (zamiast nawiasów {} jak np. w C++ czy Java). Oto kilka zasad formatowania, które należy zapamiętać:

**Wcięcia:**

* Python nie używa nawiasów {} do oznaczania bloków kodu. Zamiast tego stosuje **wcięcia** (4 spacje).
* Kod źle sformatowany może powodować błędy składniowe.

**Spójność wcięć:**

* Nie mieszaj **spacji** i **tabulatorów** – najlepiej używać 4 spacji jako wcięcie.

**Długość linii kodu:**

* Maksymalna długość linii według konwencji **79 znaków.**
* Jeśli kod jest długi, można go podzielić.

**Spacje wokół operatorów:**

* Stosowanie zapisu *a = b + c* zamiast *a=b+c.*

**Czytelność:**

* Oddzielaj bloki kodu pustymi liniami.

Plik języka Python .py z poprawnymi wcięciami

|  |
| --- |
| [module\_1\_lesson\_1\_example.py] |

|  |  |
| --- | --- |
|  | Konstrukcja *if \_\_name\_\_ == "\_\_main\_\_":* sprawia, że skrypt może być uruchamiany jako główny program lub importowany jako moduł bez jego automatycznego wykonywania. Rola tego zapisu zostanie omówiona w dalszych rozdziałach. |

Skrypt języka python z załamianiami linii

|  |
| --- |
| [module\_1\_lesson\_1\_long\_function.py] |

### Pierwsze skrypty w edytorze tekstowym

Skrypty języka Python to pliki tekstowe, więc można je edytować w dowolnym edytorze, nawet w systemowym Notatniku. Aby utworzyć pierwszy skrypt, wystarczy otworzyć Notatnik, wpisać kod w języku Python, a następnie zapisać plik z rozszerzeniem *.py*, wybierając opcję „Wszystkie pliki” w oknie zapisu. Po zapisaniu pliku można go uruchomić w konsoli systemowej, aby sprawdzić jego działanie.

Aby utworzyć nowy skrypt, w Notatniku wybierz nowy plik tekstowy i uzupełnij go treścią kodu zawartego poniżej. Następnie zapisz plik na Pulpicie z rozszerzeniem *.py*.

Program wysypujący „Witaj w swiecie Pythona”

|  |
| --- |
| [module\_1\_lesson\_1\_hello\_python.py] |

W ten sposób powstał gotowy plik z skryptem Python, który można uruchomić w konsoli systemowej. W dalszej części omówimy sposób uruchomienia tego programu.

### Uruchamianie skryptów Python w konsoli systemowej

Zanim zaczniemy korzystać z wygodnych edytorów i środowisk programistycznych (IDE), warto zrozumieć, jak uruchamiane są skrypty Python bezpośrednio w konsoli systemowej. Nowoczesne edytory, takie jak PyCharm czy VS Code, automatyzują wiele czynności, które oszczędzają czas programisty. Jednak w sytuacjach, gdy te narzędzia zawodzą lub nie są dostępne, umiejętność samodzielnego uruchamiania skryptów w terminalu może okazać się nieoceniona.

Aby uruchomić skrypt, otwórz Wiersz polecenia (CMD) (użyj skrótu Win + R, wpisz *cmd*, naciśnij Enter). Następnie przejdź do katalogu, w którym zapisano plik, np. jeśli plik jest na Pulpicie, komenda będzie podobna do poniższej.

Przejście do katalogu Pulpit w wierszu polecenia systemu Windows

|  |
| --- |
| cd C:\Users\TwojaNazwa\Pulpit |

Podczas pracy nad kodem wielokrotnie napotkamy sytuacje, w których konieczne będzie dostarczenie dodatkowych informacji do funkcji, metod czy skryptów. Takie informacje przekazujemy za pomocą argumentów. Argumenty pozwalają na dynamiczne sterowanie działaniem kodu, unikając konieczności ręcznej modyfikacji jego wnętrza.

Kod umożliwiający przekazywanie argumentów w wierszu poleceń

|  |
| --- |
| [module\_1\_lesson\_1\_arguments.py] |

### Wybór trybu pracy: interaktywny czy skryptowy?

W rozdziale wcześniejszym omówiliśmy wady i zalety korzystania z trybu interaktywnego oraz skryptowego. Teraz przeanalizujemy praktyczne zastosowania obu trybów oraz przedstawimy sytuacje, w których wybór właściwego trybu jest szczególnie istotny.

Tryb interaktywny (REPL – Read-Eval-Print Loop) jest idealnym rozwiązaniem do szybkiego testowania kodu, analizowania wyników i eksploracji bibliotek. Rozważmy teraz kilka przypadków, w których ten tryb znajdzie zastosowanie.

**Sprawdzania działania prostych komend**

W przypadkach, gdy nie ma potrzeby tworzenia całego skryptu, warto najpierw sprawdzić poprawność pojedynczej instrukcji.



Sprawdzanie działania prostych poleceń

**Eksploracji modułów i funkcji**

Tryb interaktywny umożliwia szybkie testowanie działania modułów i funkcji. Jest to wygodna i szybka metoda pozwalająca na natychmiastowe uzyskanie wyników bez konieczności tworzenia i uruchamiania osobnego skryptu lub całego projektu.



Szybka eksploracja modułów i funkcji

Tryb skryptowy pozwala na zapisanie kodu w pliku *.py*, co umożliwia jego wielokrotne uruchamianie bez konieczności każdorazowego wpisywania poleceń. Szczególna przydatność tego podejścia ujawnia się przy tworzeniu większych programów, automatyzacji zadań oraz pracy z danymi. Poniżej przedstawiono konkretne sytuacje, w których warto korzystać z trybu skryptowego.

**Tworzenie powtarzalnych skryptów**

Jeżeli kod ma być często wykonywany, zapisanie go w pliku *.py* pozwala na łatwiejsze zarządzanie i ponowne uruchamianie.

Prosty kod, który można uruchamiać wielokrotnie

|  |
| --- |
| [module\_1\_lesson\_1\_hello.py] |

**Automatyzacja zadań – przykład sprawdzania daty codziennie**

Tryb skryptowy jest szczególnie przydatny do automatyzacji powtarzalnych czynności, np. zapisywania bieżącej daty w pliku każdego dnia. Za każdym razem, gdy skrypt zostanie uruchomiony, doda nową linię do pliku *data\_log.txt* z aktualną datą i godziną.

Kod automatyzujący zapis aktualnej daty i godziny

|  |
| --- |
| [module\_1\_lesson\_1\_today\_date.py] |

Kiedy używać trybu interaktywnego, a kiedy skryptowego?

|  |  |  |
| --- | --- | --- |
| **Sytuacja** | **Tryb interaktywny** | **Tryb skryptowy** |
| Szybkie testowanie kodu | Natychmiastowe | Po zapisaniu pliku |
| Eksploracja bibliotek | Testowanie krótkich poleceń | Tworzenie większych programów |
| Tworzenie większych programów | ❌ | ✔️ |
| Automatyzacja zadań | ✔️ | ✔️ |
| Debugowanie i analiza wartości | ✔️ | ✔️ |

Podsumowując, tryb skryptowy pozwala na zapisanie kodu w pliku *.py*, co umożliwia jego wielokrotne uruchamianie i łatwiejsze zarządzanie programami. Jest niezbędny przy tworzeniu większych aplikacji oraz automatyzacji powtarzalnych zadań, takich jak przetwarzanie danych czy generowanie raportów. Dzięki obsłudze argumentów wiersza poleceń skrypty stają się bardziej elastyczne i mogą być dostosowywane do różnych scenariuszy bez konieczności modyfikacji kodu.

### Zadania

**Zadanie 1.2.1. Domyślna wersja**

* Sprawdź wersje domyślnej instancji Python.

**Zadanie 1.2.2 Tworzenie środowiska wirtualnego**

* Utwórz folder na pulpicie nazwany swoim imieniem i utwórz w nim środowisko wirtualne o tej samej nazwie.

**Zadanie 1.2.3. Tworzenie i uruchamianie prostego skryptu**

* Otwórz Notatnik i utwórz nowy plik tekstowy.
* Wpisz kod, który wypisze na ekranie tekst: "To mój pierwszy skrypt w Pythonie!".
* Zapisz plik jako pierwszy\_skrypt.py na Pulpicie.
* Otwórz Wiersz polecenia (CMD), przejdź do katalogu Pulpit i uruchom skrypt.

**Zadanie 1.2.4 \* Izolowane środowiska Python**

* Stwórz środowisko wirtualne języka Python za pomocą modułu innego niż *venv* (wykorzystaj na przykład *virtualenv*)*.*

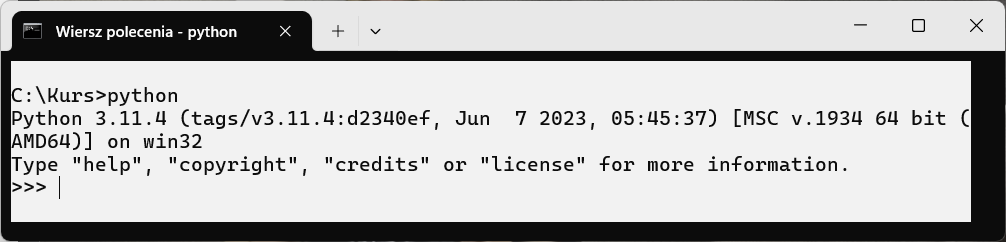
**Zadanie 1.2.5. \* Testowanie trybu interaktywnego**

* Napisz skrypt .py, który wykrywa wersję interpretera i w zależności od niej wykonuje inne operacje (np. dla Pythona <3.9 zwraca "Stara wersja", a dla >=3.9 "Nowa wersja").

## Uruchamianie kodu w konsoli

### Praca z interaktywną konsolą Python

Aby uruchomić interaktywną konsolę, wystarczy wpisać w terminalu lub wierszu poleceń: *python*. Jeśli zobaczysz znak zachęty ***>>>,*** oznacza to, że konsola jest gotowa do przyjmowania poleceń i natychmiastowego wykonywania kodu. Możesz teraz wpisywać instrukcje, które zostaną od razu przetworzone i zwrócą wynik.



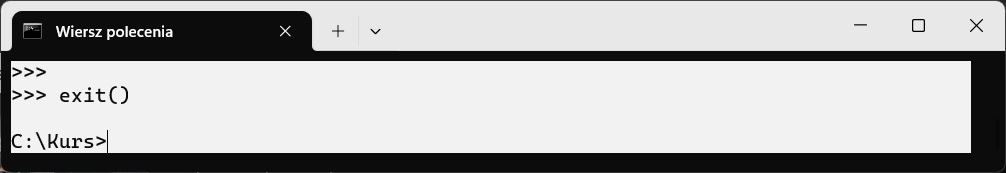
Uruchomienie trybu interaktywnego dla domyślnej wersji Python

Jeśli chcemy uruchomić konkretną wersję interpretera, musimy sprawdzić, jakie wersje są dostępne na naszym systemie. W systemie Windows można to zrobić za pomocą polecenia *py –list.* Następnie wystarczy dodać jej numer do polecenia *py*.

. Uruchomienie Python 3.10

|  |
| --- |
| py -3.10 |

Aby wyłączyć pracę w trybie interaktywnym, należy użyć komendy *exit()* lub użyć skrótuCtrl + Z (Windows) / Ctrl + D (Ubuntu). Po wykonaniu tej operacji wrócimy do wiersza poleceń (CMD) lub terminala, skąd możemy ponownie uruchomić interpreter, wykonać inne polecenia systemowe lub zamknąć okno konsoli.

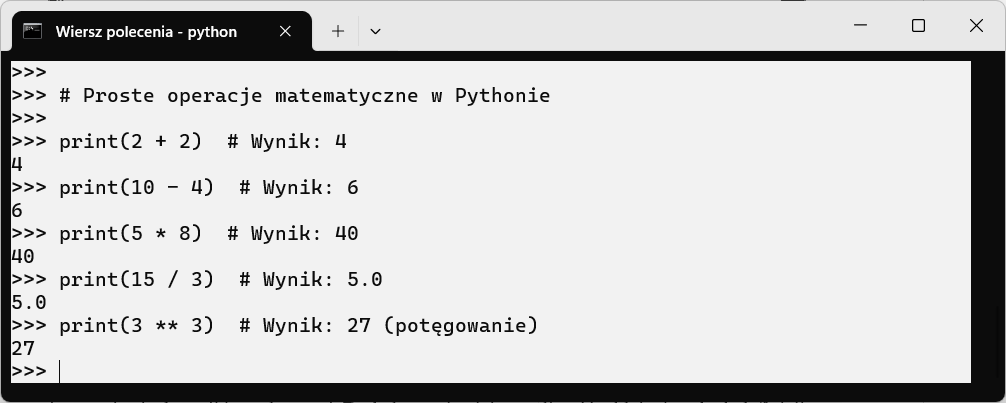


Wyłączenie trybu interaktywnego

### Wprowadzanie pojedynczych komend i natychmiastowe obserwowanie wyników

Tryb interaktywny pozwala na szybkie wykonywanie pojedynczych komend i natychmiastowe sprawdzanie ich wyników. Dzięki temu jest idealnym narzędziem do testowania kodu, eksplorowania nowych funkcji oraz nauki podstaw programowania.

W trybie interaktywnym można bezpośrednio wykonywać operacje matematyczne, przypisywać wartości do zmiennych oraz testować funkcje. W uruchomionej powłoce wystarczy wpisać 2 + 2, aby otrzymać natychmiastowy wynik. Tak samo jest z innymi, bardziej zaawansowanymi operacjami.



Natychmiastowe obserwowanie wyników w trybie interaktywnym

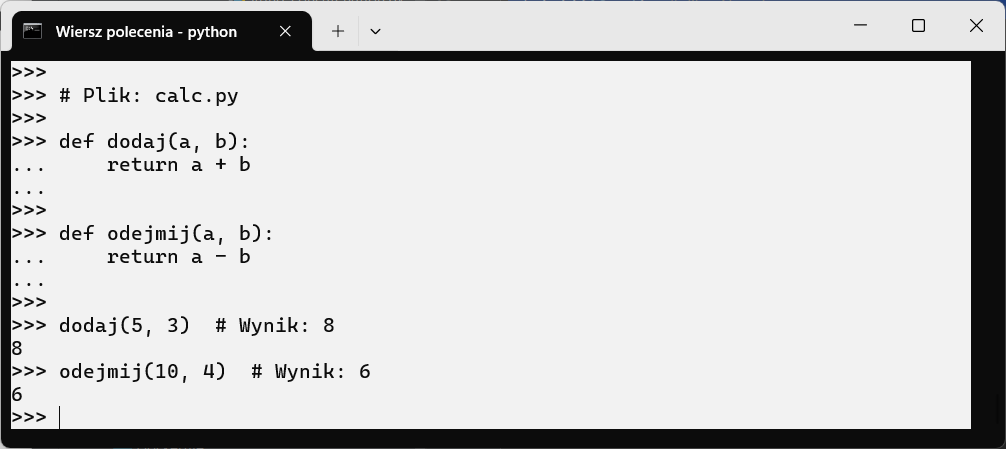
### Testowanie fragmentów kodu w trybie interaktywnym

Tryb interaktywny umożliwia nie tylko wykonywanie pojedynczych poleceń, ale także testowanie krótkich fragmentów kodu, takich jak funkcje czy klasy, które omówimy w dalszej części E-Skryptu. Można w nim również testować funkcje zapisane w osobnych plikach *.py*, importując je do sesji interaktywnej. Dodatkowo, istnieje możliwość wklejania całych definicji funkcji i wykorzystywania ich w kolejnych poleceniach, co ułatwia szybkie sprawdzanie działania kodu bez konieczności uruchamiania całego skryptu.

Wielolinijkowy fragment kodu

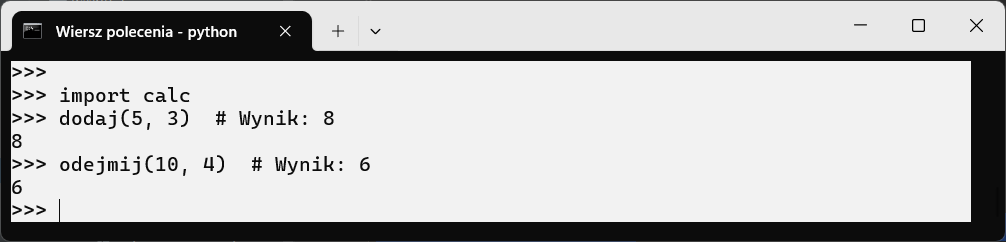
|  |
| --- |
| [module\_1\_lesson\_1\_calc.py] |

W trybie interaktywnym możemy ręcznie wkleić i uruchomić wielolinijkowy kod. Mimo, że czytelność w konsoli nie jest tak dobra jak w edytorach, to pozwala na szybkie testowanie krótkich fragmentów kodu, bez zbędnego zapisywania plików i konfiguracji środowiska.



Wielolinijkowy fragment kodu w trybie interaktywnym

Ten sam fragment kodu może zostać uruchomiony w trybie interaktywnym, ale za pomocą mechanizmów importu. To podejście jest szczególnie użyteczne w przypadkach, gdy istnieje potrzeba szybkiego uruchomienia jedynie fragmentu większego projektu.



Testowanie fragmentów kodu z pliku .py w konsoli

### Podstawowe operacje arytmetyczne w języku Python

Podstawowe operacje arytmetyczne w języku Python pozwalają na wykonywanie działań matematycznych, takich jak dodawanie, odejmowanie, mnożenie, dzielenie i potęgowanie. Dzięki prostemu składniowo systemowi operatorów, Python umożliwia szybkie i intuicyjne wykonywanie obliczeń zarówno na liczbach całkowitych, jak i zmiennoprzecinkowych.

Podstawowe operacje arytmetyczne w języku Python

|  |  |  |  |
| --- | --- | --- | --- |
| **Operacja** | **Symbol** | **Przykład** | **Wynik** |
| Dodawanie | + | 5 + 1 | 6 |
| Odejmowanie | - | 10 – 1 | 9 |
| Mnożenie | \* | 5 \* 2 | 10 |
| Dzielenie | / | 10 / 3 | 3.3333 (zwraca float) |
| Dzielenie całkowite | // | 10 // 3 | 3 (bez reszty) |
| Modulo (reszta z dzielenia) | % | 10 % 3 | 1 |
| Potęgowanie | \*\* | 2 \*\* 3 | 8 |

|  |  |
| --- | --- |
|  | Operacja dzielenia zawsze zwraca liczbę zmiennoprzecinkową typu *float*, nawet jeśli wynik jest całkowity. Z kolei dzielenie całkowite nie zapisuje części ułamkowej całkowicie ją odrzucając. |

Przykłady wykorzystania operatorów arytmetycznych przedstawiono w Błąd! Nie można odnaleźć źródła odwołania., gdzie zaprezentowano ich praktyczne zastosowanie w języku Python.

Przykłady podstawowych operacji arytmetycznych

|  |
| --- |
| [module\_1\_lesson\_1\_arithmetic.py] |

**Priorytet operatorów matematycznych**

W języku Python operatory arytmetyczne mają określony priorytet, który decyduje o kolejności wykonywania działań – najwyższy ma potęgowanie (\*\*), następnie mnożenie (\*), dzielenie (/, //, %), a najniższy dodawanie (+) i odejmowanie (-). Nawiasy () mają najwyższy priorytet i pozwalają zmieniać domyślną kolejność obliczeń.

Przykład użycia operatorów skróconych dodawania oraz potęgowania

|  |
| --- |
| [module\_1\_lesson\_1\_abbreviated\_operators.py] |

**Operacje matematyczne na różnych typach danych**

Python automatycznie konwertuje typy danych podczas operacji matematycznych, umożliwiając swobodne łączenie liczb całkowitych *int* i zmiennoprzecinkowych *float*. Przykład pokazuje, że dzielenie zawsze zwraca wartość *float*, a operacje między różnymi typami zachowują poprawność obliczeń.

Operacje matematyczne na liczbach całkowitych i zmiennoprzecinkowych

|  |
| --- |
| [module\_1\_lesson\_1\_math\_operations\_types.py] |

### Manipulacja napisami

Napisy (łańcuchy znaków) w języku Python pozwalają na różnorodne operacje, takie jak łączenie operatorem +, powielanie za pomocą operatora \* oraz zmiana wielkości liter. Dzięki wbudowanym metodom i operatorom język umożliwia łatwe przetwarzanie tekstu, co jest szczególnie przydatne w pracy z danymi tekstowymi.

**Łączenie napisów (konkatenacja)**

Łączenie napisów, czyli konkatenacja, pozwala na scalanie dwóch lub więcej ciągów znaków w jeden. W języku Python można to zrobić za pomocą operatora *+, join(), f-stringów* lub metody *format()*.

Różne metody łączenia napisów

|  |  |  |  |
| --- | --- | --- | --- |
| **Metoda** | **Opis** | **Przykład** | **Wynik** |
| + | Łączy napisy w prosty sposób | "Hello" + " World" | "Hello World" |
| join() | Łączy elementy listy w jeden ciąg znaków | " ".join(["Hello", "World"]) | "Hello World" |
| f-string | Dynamiczne formatowanie napisów | f"Hi {name}" (name = "John") | "Hi John" |
| .format() | Alternatywna metoda formatowania | " Hi {}".format("John") | " Hi John" |

. Przykłady użycia metod łączenia napisów

|  |  |
| --- | --- |
| [module\_1\_lesson\_1\_string\_concatenation.py] |  |

**Powielanie napisów**

Powielanie napisów pozwala na wielokrotne powtórzenie tego samego ciągu znaków. Można to zrobić za pomocą operatora \*, który multiplikuje napis przez podaną liczbę.

Powielanie napisów

|  |  |  |  |
| --- | --- | --- | --- |
| **Metoda** | **Opis** | **Przykład** | **Wynik** |
| \* | Powiela napis określoną liczbę razy | "Ha " \* 3 | "Ha Ha Ha" |
| join() | Powiela element w kontekście listy | "-".join(["Ha"] \* 3) | "Ha-Ha-Ha" |

. Przykład użycia powielania napisów

|  |
| --- |
| [module\_1\_lesson\_1\_string\_repetition.py] |

**Zmiana wielkości liter**

Python oferuje wbudowane metody do zmiany wielkości liter w napisach. Poniższa tabela przedstawia wbudowane metody Pythona umożliwiające zmianę wielkości liter w napisach, takie jak *upper()*, *lower()*, *title()* i *capitalize()*. Dzięki nim można łatwo konwertować tekst do wielkich lub małych liter oraz formatować nagłówki i zdania zgodnie z wymaganiami.

Metody zmiany wielkości liter w języku Python

|  |  |  |  |
| --- | --- | --- | --- |
| **Metoda** | **Opis** | **Przykład wejściowy** | **Wynik** |
| .upper() | Zamienia wszystkie litery na wielkie | "python jest świetny!" | "PYTHON JEST ŚWIETNY!" |
| .lower() | Zamienia wszystkie litery na małe | "Python JEST Świetny!" | "python jest świetny!" |
| .title() | Pierwsza litera każdego słowa jest wielka | "python jest świetny!" | "Python Jest Świetny!" |
| .capitalize() | Pierwsza litera napisu jest wielka, reszta mała | "python JEST świetny!" | "Python jest świetny!" |

Przykład użycia metod zmiany wielkości liter

|  |
| --- |
| [module\_1\_lesson\_1\_text\_case\_methods.py] |

### Wykorzystanie wbudowanych funkcji: len() i type()

Python oferuje wiele wbudowanych funkcji, które ułatwiają pracę z danymi. Dwie z nich, *len()* i *type()*, są niezwykle przydatne podczas analizy zmiennych i manipulacji danymi.

Wykorzystanie funkcji wbudowanych len() i type()

|  |  |  |  |
| --- | --- | --- | --- |
| **Funkcja** | **Opis** | **Przykład** | **Wynik** |
| len() | Zwraca liczbę znaków w napisie | len("Python") | 6 |
| len() | Zwraca liczbę elementów w liście | len([1, 2, 3, 4]) | 4 |
| len() | Liczy klucze w słowniku | len({"a": 1, "b": 2}) | 2 |
| type() | Sprawdza typ zmiennej (liczba całkowita) | type(42) | <class 'int'> |
| type() | Sprawdza typ zmiennej (tekst) | type("Hello") | <class 'str'> |
| type() | Sprawdza typ zmiennej (lista) | type([1, 2, 3]) | <class 'list'> |
| type() | Sprawdza typ zmiennej (liczba całkowita) | type(42) | <class 'int'> |

Dzięki funkcjom *len()* i *type()* można w łatwy sposób analizować dane i dynamicznie sprawdzać ich właściwości. Funkcja *len()* pozwala określić liczbę znaków w napisie lub liczbę elementów w kolekcjach, takich jak listy czy słowniki, natomiast *type()* umożliwia sprawdzenie typu zmiennej, co jest przydatne podczas debugowania i pracy z różnymi typami danych.

Działanie funkcji *len()* na różnych typach danych

|  |
| --- |
| [module\_1\_lesson\_1\_len\_type\_demo.py] |

### Zadania

**Zadanie 1.3.1. Uruchomienie wybranej wersji Python**

* Otwórz systemowy wiersz poleceń (CMD).
* Wyświetl w konsoli listę dostępnych wersji Python.
* Uruchom Python dla domyślnej wersji Python (oznaczonej gwiazdką na liście).
* Nie stosuj komendy *python,* zamiast niej użyj *py* z odpowiednim argumentem.

**Zadanie 1.3.2. Testowanie trybu interaktywnego**

* Otwórz Wiersz polecenia (CMD) i uruchom tryb interaktywny wpisując *python*.
* Wykonaj operacje mnożenia dwóch liczb (np. *2\*2*), wypisz swoje imię funkcją *print().*
* Zakończ tryb interaktywny poleceniem *exit()* lub kombinacją klawiszy *Ctrl + Z* i *Enter*.

**Zadanie 1.3.3. Łączenie napisów**

Napisz program, który połączy napisy “Jan” oraz “Kowalski” za pomocą operatora +.

**Zadanie 1.3.4. \* Obsługa argumentów w skrypcie Pythona**

* Napisz skrypt, który przyjmuje wiele argumentów i wypisuje ich listę w formie:
  + Podano X argumentów: *Tu lista argumentów.*
* Uruchom ten kod z poziomu trybu interaktywnego z argumentami.

**Zadanie 1.3.5. \* Wielkość liter**

* Wyświetli “Jan Kowalski” w trzech formatach oraz wypisz typ zmiennej i długość:
  + **Za pomocą +**: JanKowalski
  + **Za pomocą *join()***: jan kowalski
  + **Wielkimi literami:** JANKOWALSKI

## Typy podstawowe oraz metody wczytywania danych

W poniższym rozdziale przybliżone zostaną typy występujące w języku Python i metody za pomocą których można wczytać dane od użytkownika.

### Typy liczbowe

Typy liczbowe stanowią podstawową kategorię danych służących do reprezentacji wartości liczbowych. W standardowej dystrybucji języka Python wyróżnia się trzy główne typy liczbowe: całkowite, zmiennoprzecinkowe oraz zespolone.

**Liczby całkowite (int)**

Obejmują wartości bez części dziesiętnej, mogą być dodatnie, ujemne lub zerowe, a ich precyzja nie jest ograniczona przez stałą liczbę bitów – wielkość liczby jest ograniczona jedynie przez dostępną pamięć.



Przykłady liczb całkowitych

**Liczby zmiennoprzecinkowe (float)**

Reprezentują liczby rzeczywiste, czyli wartości zawierające część ułamkową. Reprezentacja ta opiera się na standardzie IEEE 754, co wiąże się z ograniczeniami precyzji oraz możliwością wystąpienia drobnych błędów zaokrągleń przy operacjach matematycznych.



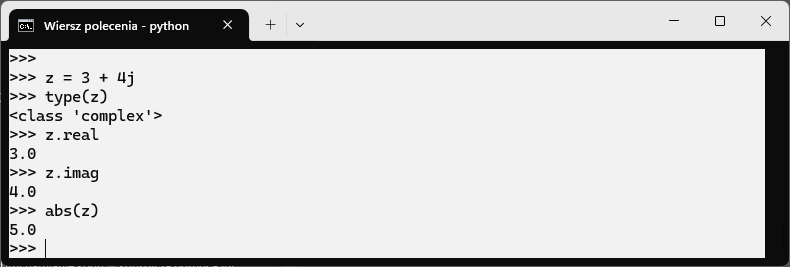
Przykłady liczb zmiennoprzecinkowych



Przykład błędu zaokrąglenia liczb zmiennoprzecinkowych

**Liczby zespolone** **(complex)**

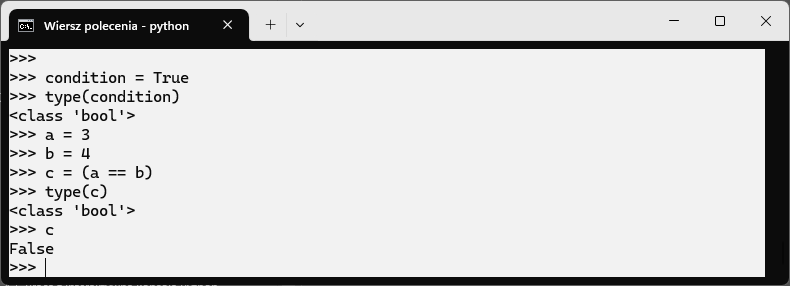
Składają się z części rzeczywistej oraz urojonej. Notacja liczby zespolonej w Pythonie wykorzystuje literę j do oznaczenia części urojonej, na przykład zapis *3 + 4j* oznacza liczbę zespoloną, w której część rzeczywista wynosi 3, a część urojona 4.



Przykład użycia liczby zespolonej

**Wartość logiczna (bool)**

Stanowi typ przeznaczony do reprezentowania wartości logicznych rozumianych jako *true* lub *false*. Wykorzystywane są głównie w operacjach warunkowych i strukturach kontrolnych, takich jak instrukcje warunkowe czy pętle. Typ *bool* jest powiązany z typem *int*, przy czym wartość *true* odpowiada liczbie 1 a *false* liczbie 0, co umożliwia pewne operacje arytmetyczne.



Przykład deklaracji typu logicznego i jego generacja na wskutek operacji porównania

### Typy sekwencyjne

Typy sekwencyjne to kategorie danych reprezentujące uporządkowane kolekcje elementów, w których kolejność występowania ma znaczenie. Umożliwiają one dostęp do poszczególnych elementów poprzez indeksowanie, operacje cięcia oraz iterację. Istnieje kilka podstawowych typów sekwencyjnych, z których każdy charakteryzuje się określonymi właściwościami i sposobem przechowywania danych.

**Łańcuch znaków (string)**

Obiekty typu *str*, oznaczone jako *str*, są sekwencjami znaków i charakteryzują się niemutowalnością – po ich utworzeniu zawartość nie może być zmieniona, a każda operacja modyfikująca ciąg znaków skutkuje utworzeniem nowego obiektu.

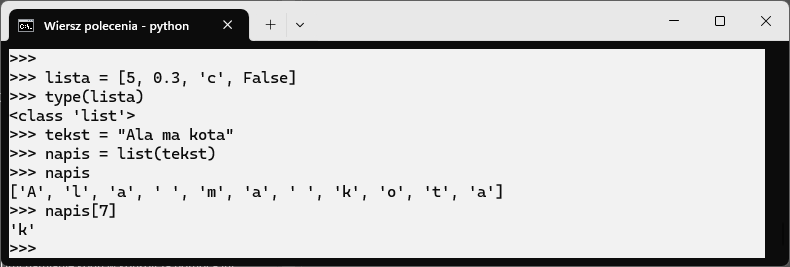


Przykład stworzenia ciągu znaków w postaci zdania oraz pojedynczego znaku

|  |  |
| --- | --- |
|  | Kiedy mówimy, że zmienne nie mogą być zmieniane, nazywamy to *niemutowalnością*. |

**Listy (list)**

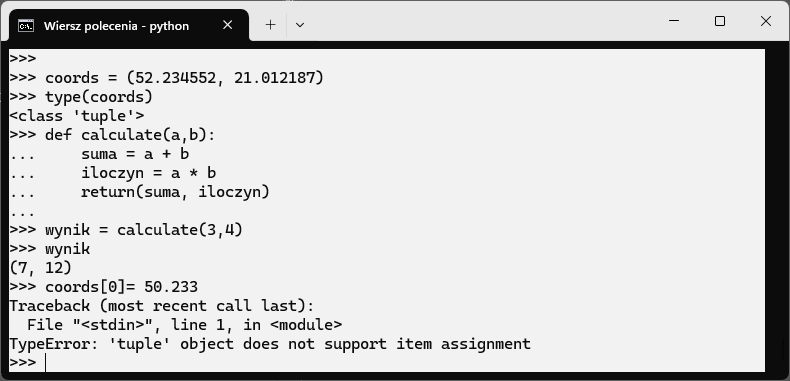
Stanowią mutowalną sekwencję, co oznacza, że zawartość listy może być modyfikowana po jej utworzeniu. W liście można przechowywać elementy różnych typów, a ich kolejność jest zachowywana.



Przykład utworzenia listy zawierającej wiele typów danych

**Krotki (tuple)**

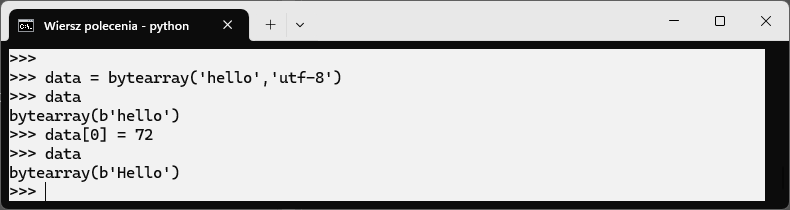
Krotki to sekwencje, podobne do list, ale raz utworzone nie można ich zmienić. Oznacza to, że dane zapisane w krotce pozostają stałe, co jest przydatne, gdy potrzebujemy pewności, że zestaw informacji nie zostanie przypadkowo zmodyfikowany.



Niezmienność krotek w języku Python

**Sekwencja bajtów (bytes/bytearray)**

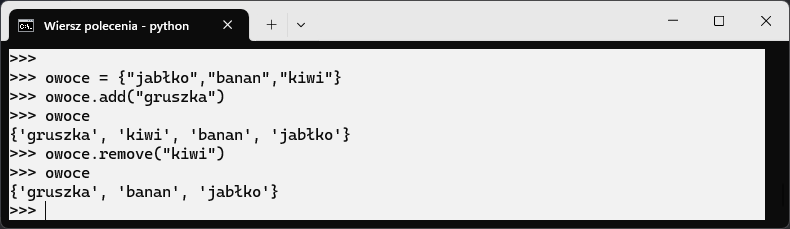
Sekwencyjne typy reprezentujące dane binarne. Typ *bytes* jest niemutowalnym ciągiem liczb całkowitych, gdzie każda liczba mieści się w przedziale od 0 do 255, i jest wykorzystywany do reprezentowania danych binarnych. Wersja mutowalna, *bytearray*, umożliwia modyfikację zawartości po jej utworzeniu, co jest przydatne w operacjach na danych niskopoziomowych.



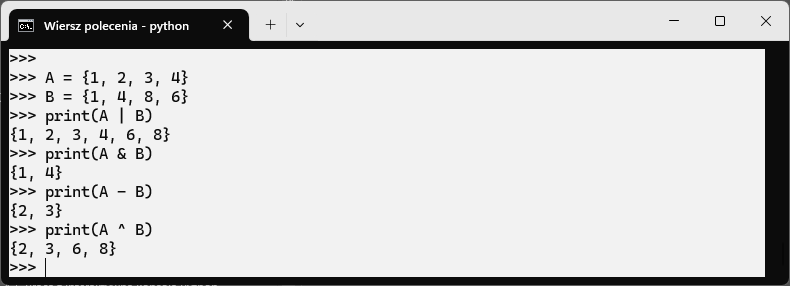
Przykład utworzenia krotki oraz stowrzenia jej jako zwracany typ funkcji

### Zbiory

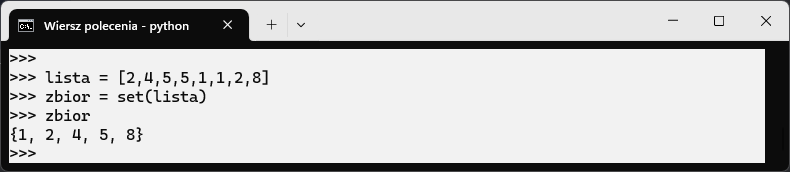
Zbiór tworzony jest poprzez ujęcie elementów w nawiasy klamrowe, na przykład: *{1, 2, 3},* lub za pomocą funkcji *set(),* która konwertuje iterowalne obiekty na zbiór. Zbiór umożliwia wykonywanie operacji matematycznych, takich jak suma, różnica, przecięcie czy symetryczna różnica, co przydaje się przy analizie zbiorów danych oraz usuwaniu duplikatów.



Operacja utworzenia zbioru i dodania i usunięcia jego elementu



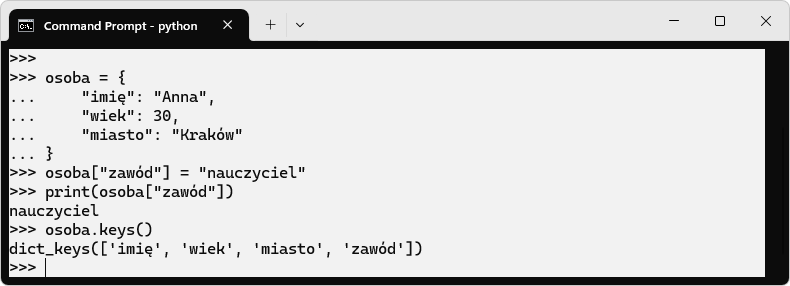
Podstawowe operacje na zbiorach



Usuwanie duplikatów i sortowanie elementów listy poprzez rzutowanie

### Typ mapujący

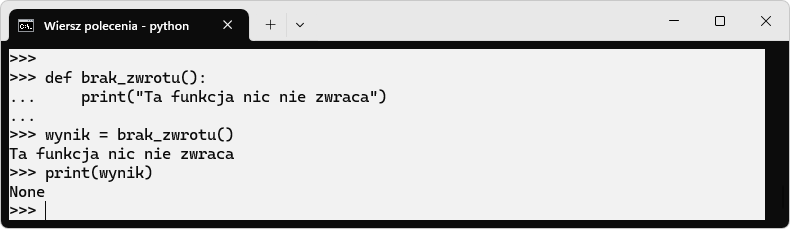
Słownik (dictionary) to jeden z typów mapujących języka Python, oparty na unikalnych kluczach niemutowanego typu, które są przypisane do odpowiadających im wartości. Elementy słownika zapisuje się w nawiasach klamrowych w formie par klucz-wartość, oddzielonych dwukropkiem. Słowniki umożliwiają szybki dostęp do danych oraz dynamiczne dodawanie, modyfikowanie i usuwanie elementów, co czyni je wszechstronnym narzędziem do przetwarzania informacji.



Utworzenie słownika, dodanie pola "zawód" i wyświetlenie kluczy jakie zawiera

### Typ braku wartości

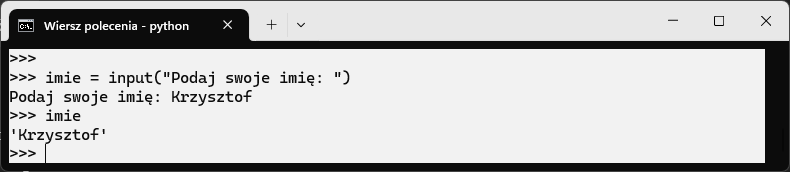
Wbudowany typ danych w *NoneType* reprezentuje brak wartości. Obiekt *None* pełni funkcję domyślnego wyniku funkcji, które nie zwracają jawnie innej wartości, oraz wskazuje na niezainicjowaną zmienną. Dodatkowo zabezpiecza aplikację przed nieprawidłowym dostępem do pamięci, chroniąc ją przed potencjalnymi błędami.



Zwrócenie wartości None dla funkcji bez zwracanego typu

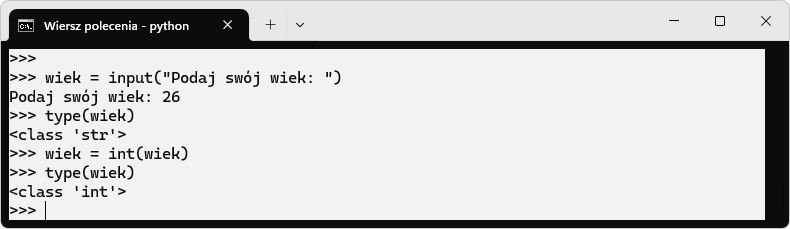
### Funkcja input()

Funkcja *input()* umożliwia pobieranie danych od użytkownika poprzez standardowe wejście (klawiaturę). Podczas wywołania funkcji program oczekuje na wpisanie tekstu, a następnie zwraca wprowadzony ciąg znaków. Argument przekazywany do funkcji *input()* stanowi komunikat wyświetlany użytkownikowi, informujący o oczekiwanym rodzaju danych. Na przykład, wywołanie zaprezentowane poniżej, spowoduje, że na ekranie pojawi się komunikat „Podaj swoje imię:”, a użytkownik, po wpisaniu swojego imienia i naciśnięciu klawisza Enter, zapisze wartość w zmiennej *imie.*



Pobranie od użytkownika danych za pomocą funkcji input()

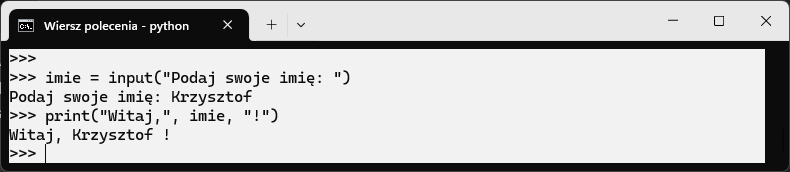
Warto zauważyć, że funkcja *input()* zawsze zwraca wartość typu string, co oznacza, że przy pobieraniu liczb konieczna jest konwersja, na przykład przy użyciu funkcji *int()* lub *float()*:



Pobranie informacji od użytkownika i konwersja jej typu z str na int

### Funkcja print()

Funkcja *print()* służy do prezentowania wyników na ekranie poprzez standardowe wyjście. Argumenty przekazywane do funkcji są konwertowane na ciągi znaków, a następnie wyświetlane. Domyślnie kolejne argumenty oddzielane są spacją, a na końcu wyniku funkcja dodaje znak nowej linii. Przykładowe użycie funkcji *print()* połączy zawartość zmiennej z innymi fragmentami tekstu i wyświetli komunikat. Funkcja *print()* pozwala także na drukowanie wyników obliczeń, wartości zmiennych lub złożonych tekstów.



Pobranie informacji od użytkownika i wyświetlenie jej za pomocą funkcji print()

### Zadania

**Zadanie 1.4.1. Sprawdzenie typów utworzonych zmiennych**

Utwórz zmieną x = tuple([1, 2, 3]) i wypisz jej typ.

**Zadanie 1.4.2. Sumowanie różnych typów**

* Utwórz zmienne *x = 5* i *y = 3.14*
* Wykonaj operację:
  + *x + y,*  i wypisz typ.
  + *str(x) + str(y),* i wypisz typ.

**Zadanie 1.4.3. Przeliczanie temperatury**

* Stwórz program, który poprosi użytkownika o podanie temperatury w stopniach Celsjusza, a następnie obliczy i wyświetli jej odpowiednik w stopniach Fahrenheita. Użyj formuły:
* *F = (C × 9/5) + 32*
* Wynik powinien być wyświetlony w formacie: *"[C]°C to [F]°F."*

**Zadanie 1.4.4. \* Liczby zespolone**

* Napisz program, który tworzy dwie liczby zespolone, dodaje je do siebie i wypisuje wynik wraz z typem.

**Zadanie 1.4.5. \* Średnia i różnica liczb**

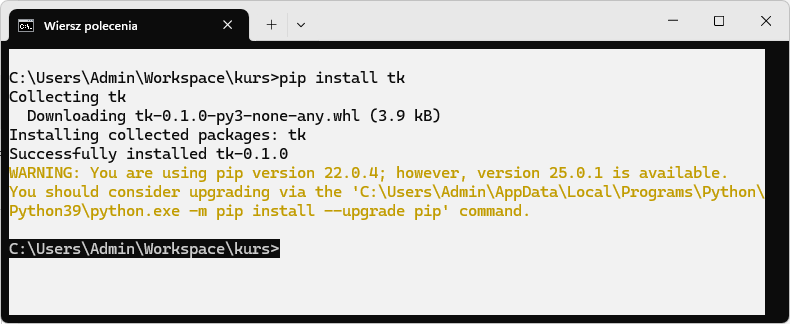
* Stwórz program, który poprosi użytkownika o trzy liczby całkowite, a następnie obliczy ich średnią oraz różnicę między największą i najmniejszą wartością. Wyniki wyświetl w formacie:
* *"Średnia z podanych liczb to: [średnia]. Różnica między największą a najmniejszą liczbą to: [różnica]."*

## Instalacja IDE oraz omówienie menedżera PIP

Zintegrowane środowisko programistyczne oraz menedżer bibliotek PIP to podstawowe narzędzia programistyczne które na co dzień są wykorzystywane przy programowaniu w języku Python. W poniższym rozdziale przedstawione zostaną sposoby jak je wykorzystać w codziennej pracy.

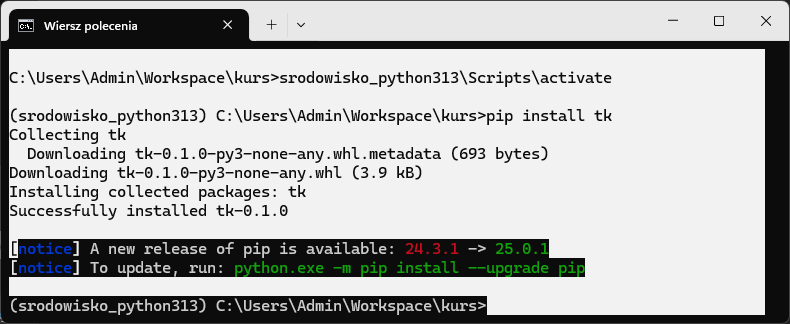
### Instalacja i wykorzystanie bibliotek z pomocą narzędzia PIP

PIP to menedżer pakietów, umożliwiający instalację oraz zarządzanie zewnętrznymi bibliotekami, które zwiększają funkcjonalność Pythona. Aby znaleźć bibliotekę, można odwiedzić stronę projektu PyPI[[2]](#footnote-3), gdzie dostępne są tysiące pakietów. Instalacja biblioteki odbywa się za pomocą polecenia *pip* wpisywanego w terminalu naszego systemu.



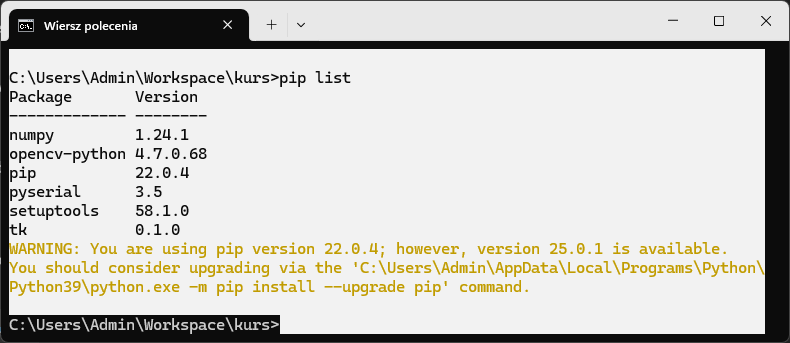
Przykład instalacji biblioteki tkinter na poziomie globalnym

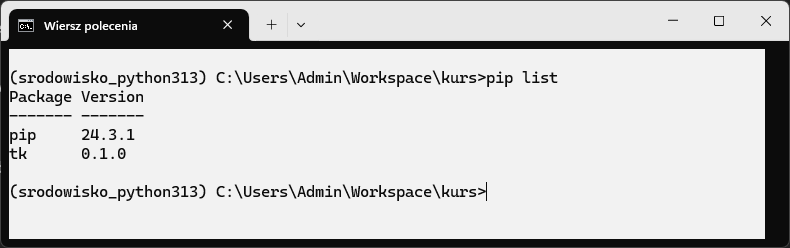
Instalowanie bibliotek za jego pomocą jest bardzo wygodne i szybkie. Należy jednak mieć na uwadze fakt, że wykonywanie go na poziomie systemu, powoduje, że biblioteka jest instalowana globalnie i może to powodować konflikty z innymi modułami korzystającymi z innych zależności. Dlatego też, przy dużych projektach wykorzystuje się wirtualne środowiska, które mają izolować od siebie zarówno wersje języka Python, ale również biblioteki, które mogą wchodzić ze sobą w konflikt.



Przykład instalacji biblioteki tkinter wewnątrz środowiska wirtualnego

Możliwe jest także wyświetlenie wszystkich zainstalowanych bibliotek, zarówno na poziomie globalnym jak i wewnątrz środowiska wirtualnego.





Przykład wyświetlenia zawartości zainstalowanych bibliotekach dla Pythona dostępnego globalnie w systemie oraz wewnątrz środowiska wirtualnego

### Instalacja i korzystanie z bibliotek na przykładzie turtle

Jedną z wielu bibliotek dostępnych w PIP jest *turtle*, która nie jest domyślnie instalowana z Pythonem. Aby z niej korzystać, należy ją zainstalować za pomocą poznanego menedżera PIP, wpisując odpowiednią komendę

Instalacja biblioteki turtle

|  |
| --- |
| pip install turtle |

Po instalacji możemy sprawdzić listę dostępnych bibliotek, aby upewnić się, że biblioteka została poprawnie dodana do projektu.

Sprawdzenie listy zainstalowanych bibliotek

|  |
| --- |
| pip list |

Tabela przedstawia podstawowe metody dostępne w bibliotece *turtle*, które służą do sterowania ruchem oraz wyglądem rysowanych kształtów.

Funkcje biblioteki turtle

|  |  |
| --- | --- |
| **Metoda** | **Znaczenie** |
| forward(x) | przesuwa żółwia do przodu o *x* pikseli |
| left(kąt) / right(kąt) | obraca żółwia o podany kąt |
| penup() / pendown() | podnosi lub opuszcza „pióro”, sterując rysowaniem |
| color(kolor) | zmienia kolor linii |
| width(grubość) | ustawia grubość linii |
| speed(prędkość) | reguluje szybkość rysowania |

Teraz narysujemy kilka prostych figur geometrycznych, aby zobaczyć, jak działa ta biblioteka i jak można ją wykorzystać do implementacji podstawowych algorytmów. Pierwszy kod rysuje kwadrat, przesuwając żółwia do przodu i obracając go o 90 stopni cztery razy. Drugi kod rysuje trójkąt, ustawiając niebieski kolor i grubość linii, a następnie przesuwając żółwia i obracając go o 120 stopni trzy razy.

Rysowanie kwadratu za pomocą biblioteki

|  |
| --- |
| [module\_1\_lesson\_2\_turtle\_square.py] |

Rysowanie trójkąta za pomocą biblioteki

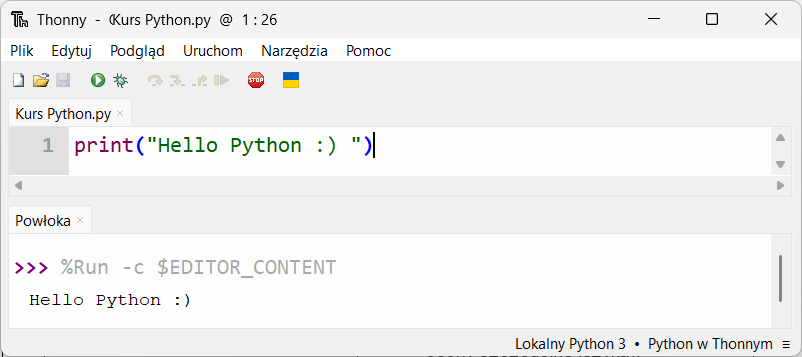
|  |
| --- |
| [module\_1\_lesson\_2\_turtle\_triangle.py] |

### Omówienie różnych IDE

W module poruszono temat kilku popularnych środowisk i edytorów tekstów, które mogą służyć pisaniu skryptów w języku Python, natomiast w niniejszym rozdziale omówimy je bardziej szczegółowo. Warto się z nimi zapoznać i stwierdzić czy odpowiadają naszym preferencjom i zadaniom jakie w nich będziemy realizować.

**Thonny**

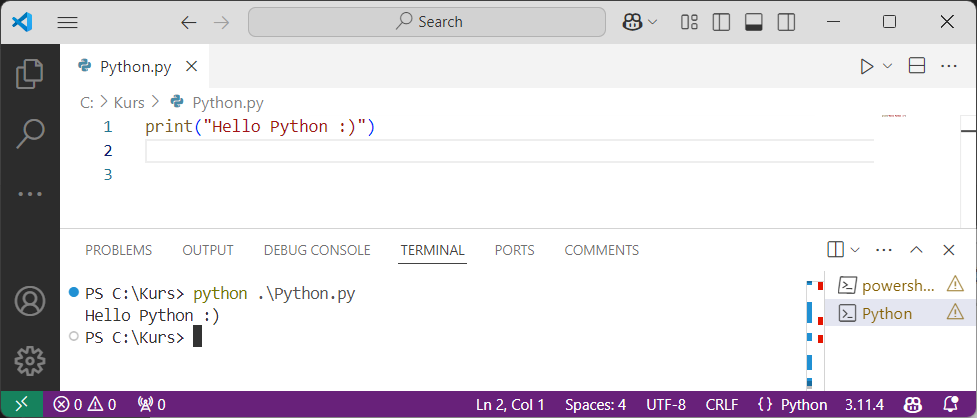
Jest to środowisko dedykowane początkującym programistom. Program działa na wszystkich popularnych systemach operacyjnych i oferuje wszystkie niezbędne narzędzia do kodowania, takie jak interpreter, debugger oraz podświetlanie błędów. Jego wymagania sprzętowe są niewielkie dzięki czemu zadziała nawet na słabszych komputerach. Jednakże, z uwagi na brak widoku drzewa folderów i integracji z narzędziami typu Git, jest on zalecany głównie do mniejszych projektów. Oprogramowanie udostępniane jest na licencji MIT.



Okno edytora Thonny

**Visual Studio Code**

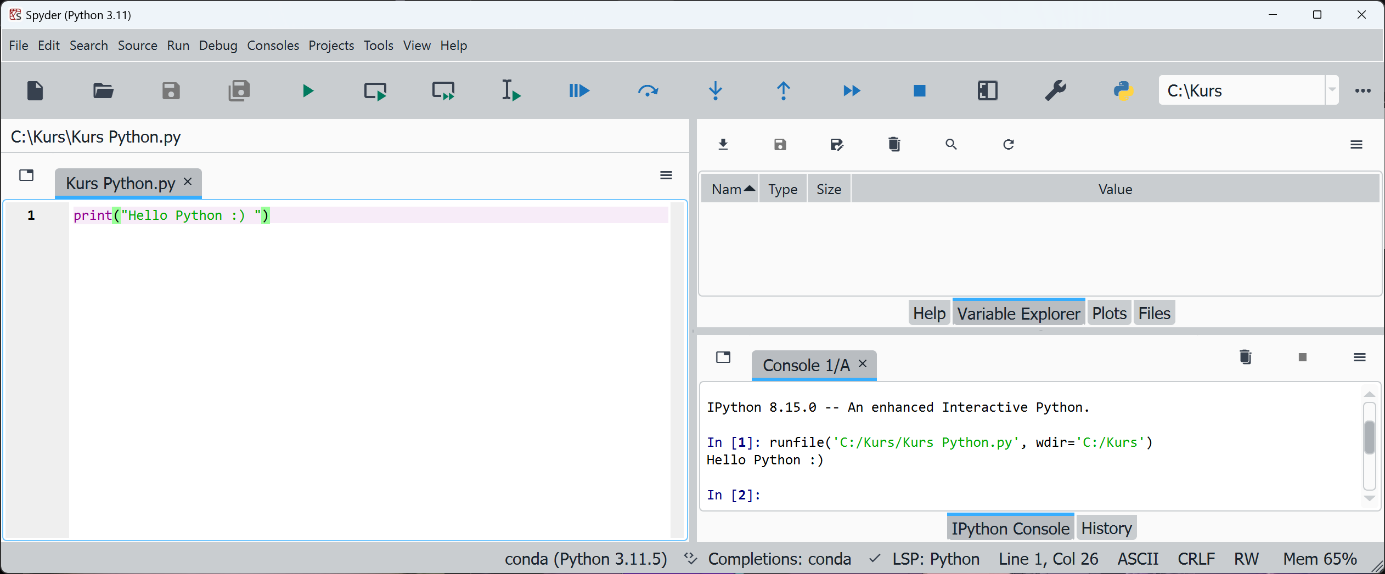
Edytor opracowany przez firmę Microsoft, udostępniany na licencji MIT, oferuje wiele możliwości. Posiada wbudowany sklep, dzięki któremu można dodawać wtyczki i dostosowywać środowisko do indywidualnych potrzeb. Jego wszechstronność umożliwia integrację z Git, prezentację drzewa folderów, obsługę debuggera oraz pracę w wielu językach. Dodatkowo oferuje funkcję podpowiadania składni. Bogactwo funkcji sprawia jednak, że konfiguracja środowiska na początek może być skomplikowana i czasochłonna, a przy dużej liczbie wtyczek i otwartych projektów edytor może wymagać większych zasobów do płynnego działania.



Okno edytora Visual Studio Code

**Spyder**

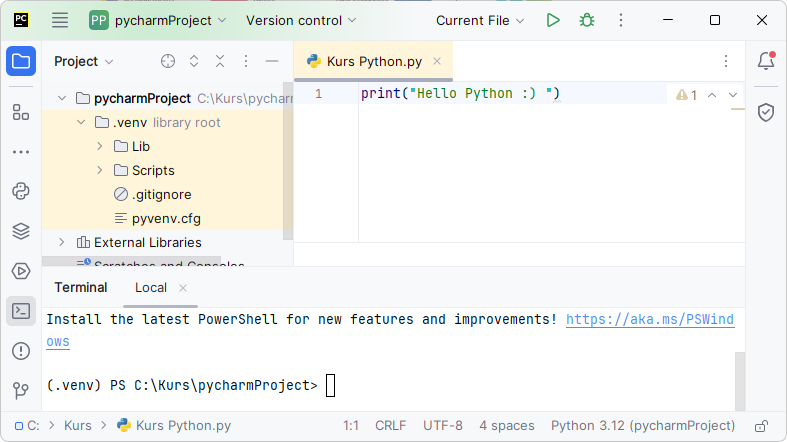
Jest to narzędzie zaprojektowane z myślą o naukowcach i osobach prowadzących symulacje oraz skomplikowane obliczenia w środowisku Python. Poza edytorem kodu oferuje dostęp do interaktywnej konsoli oraz umożliwia wyświetlanie generowanych przez kod tabel, wykresów i grafik. Ze względu na swoje specyficzne zastosowanie, może być mniej przydatne do ogólnych zadań. Ponadto wymaga większych zasobów sprzętowych niż inne IDE. Program dystrybuowany jest na licencji MIT.



Okno edytora Spyder

**PyCharm**

Jest to najbardziej zaawansowane IDE dla języka Python. Oferuje funkcje refaktoryzacji kodu, debuggera, kontroli wersji Git, podpowiedzi składni oraz wiele innych. IDE jest zintegrowane z popularnymi bibliotekami i zapewnia rozbudowane zarządzanie środowiskiem. Niestety, wiąże się to z wysokimi wymaganiami systemowymi oraz złożoną konfiguracją na początku pracy. Program dystrybuowany jest w wersji darmowej, która ma ograniczoną funkcjonalność, natomiast wersja pełna jest płatna.



Okno edytora PyCharm

### Instalacja wybranych IDE: Thonny i Visual Studio Code

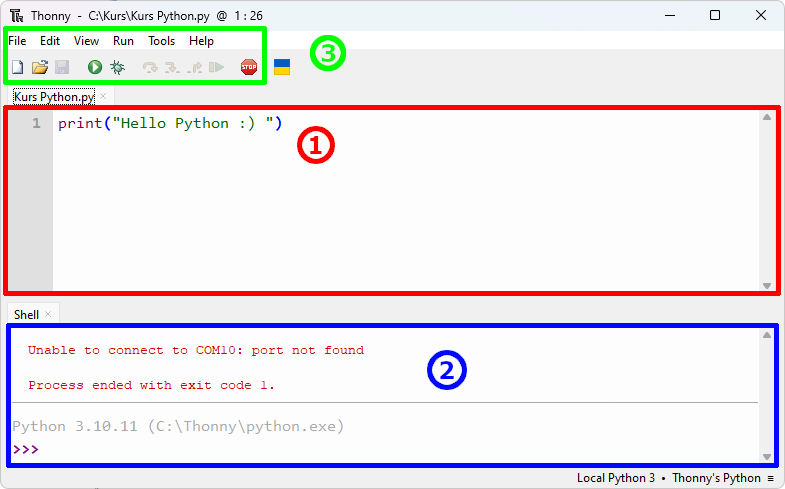
W dalszej części kursu, jako przykładowe zintegrowane środowiska programistyczne posłużą Thonny i Visual Studio Code. Aby je zainstalować na swoim komputerze, należy odwiedzić stronę projektu odpowiedniego środowiska (Thonny[[3]](#footnote-4) lub Visual Studio Code[[4]](#footnote-5)) i wybrać wersję przeznaczoną dla swojego systemu operacyjnego. W przypadku pierwszego programu dostępne są dwie opcje – instalator oraz wersja przenośna, która nie wymaga instalacji. Visual Studio Code oferuje więcej opcji instalacji, jednak każda z nich wymaga pobrania instalatora i instalacji oprogramowania na komputerze.

|  |  |
| --- | --- |
|  |  |

Okna pobrania IDE Thonny i Visual Studio Code

### Zapoznanie się z funkcjami Thonny

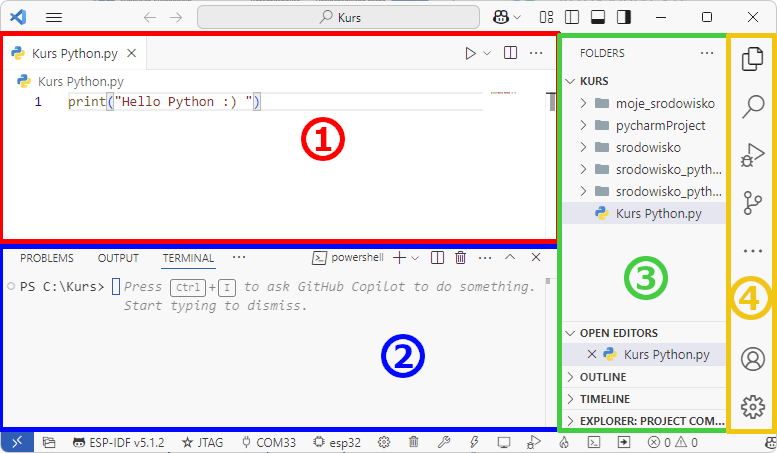
Po uruchomieniu programu wyświetla się interfejs programistyczny. W centralnej części znajduje się okno edytora kodu (1). Każdorazowo, gdy otwierany jest nowy plik, nad edytorem pojawia się zakładka z nazwą pliku, umożliwiająca płynne przełączanie się między nimi. Poniżej znajduje się interaktywny terminal (2), który pozwala testować kod w trybie interaktywnym oraz uruchamiać pliki źródłowe. Na samej górze umieszczony jest pasek narzędzi z najczęściej używanymi opcjami (3). Od lewej znajdują się przyciski otwierania plików lub folderów oraz zapisywania zmian, a obok nich opcje uruchamiania skryptu w trybie normalnym lub debugowania. W trybie usuwania błędów aktywowane są cztery strzałki, oznaczające: „przejście dalej”, „wejście” (np. do funkcji), „wyjście” oraz „zatrzymanie” lub „wznowienie” działania kodu, gdy jest on uruchamiany w pętli.



Oznaczone okno edytora kodu (1), interaktywnego terminala (2) i paska narzędzi (3) w edytorze Thonny

### Zapoznanie się z funkcjami VS Code

Po zainstalowaniu IDE przywita nas ekran powitalny z którego można wybrać opcję otwarcia folderu lub pliku z kodem źródłowym. Wybierając opcję otwarcia folderu załaduje się ekran edycji z terminalem (2), drzewem plików i folderów (3) oraz pasek aktywności (4). Aby móc edytować kod, z drzewa plików należy wybrać docelowy plik źródłowy i otworzyć go, poprzez dwukrotne naciśnięcie lewym przyciskiem myszy. Wówczas wyświetli się jego zawartość z możliwością edytowania (1). Program sam wykrywa w jakim języku zapisany jest kod i automatycznie podświetla odpowiednią składnię. W górnej części okna edytora znajduje się strzałka która uruchamia kod. Gdy wykonywane jest to po raz pierwszy po zainstalowaniu programu, ten poprosi nas o wyznaczenie za pomocą którego z wykrytych w systemie interpreterów skorzystać. Możliwe jest także samodzielne wskazanie ścieżki do niego.



Oznaczone okno edytora kodu (1), interaktywnego terminala (2), drzewa plików i folderów (3) oraz pasek aktywności (4) w edytorze Visual Studio Code

### Zadania

**Zadanie 1.5.1. Instalacja i uruchomienie IDE**

Uruchom środowisko i uruchom prosty skrypt Python: *print("Witaj w Pythonie!")*

**Zadanie 1.5.2. Instalacja dodatkowej biblioteki**

* Za pomocą menedżera PIP zainstaluj bibliotekę *requests*
* W skrypcie .py zaimportuj pobraną bibliotekę i wyświetl jej wersję

**Zadanie 1.5.3. Rysowanie gwiazdy**

* Napisz skrypt importujący bibliotekę *turtle*
* Za pomocą funkcji zaimportowanej biblioteki, narysuj pięcioramienną gwiazdę

**Zadanie 1.5.4. \* Konfiguracja debuggera w Thonny**

Uruchom Thonny i napisz program, który dodaje dwie liczby podane przez użytkownika.

Ustaw punkt przerwania (breakpoint) w linii, gdzie wykonywane jest dodawanie, a następnie przeanalizuj wartości zmiennych w trybie debugowania.

**Zadanie 1.5.5. \* Rysowanie spirali**

* Napisz skrypt z użyciem biblioteki *turtle*, który narysuje spiralę złożoną z pojedynczych linii. Użyj prostej pętli for o 20 iteracjach według poniższego algorytmu:
  + Ustal długość kroku
  + Dla każdej iteracji od 1 do 20:
    - Przesuń żółwia do przodu o ustaloną długość kroku
    - Obróć żółwia w lewo o 30 stopni
    - Zwiększ wartość kroku o 5

## Formatowanie kodu oraz korzystanie z IDE

Pisząc kod w języku Python należy zwrócić na poprawne formatowanie kodu. Ma to bezpośredni wpływ nie tylko na wygląd i zrozumienie pisanych algorytmów ale również na ich bezbłędne działanie.

### Zasady PEP 8

PEP, to formalny dokument (ang, Python Enhancement Proposal), w którym przedstawiane są propozycje zmian i ulepszeń języka Python. Dokumenty te służą jako przewodniki dla społeczności programistycznej, pomagając w standaryzacji najlepszych praktyk i kształtowaniu rozwoju języka. Najbardziej znanym przykładem jest PEP 8[[5]](#footnote-6), czyli przewodnik stylu, który określa zasady pisania czytelnego i spójnego kodu. Opisuje on między innymi:

* Wcięcia i formatowanie kodu.
* Maksymalną długości linii.
* Zasadę stosowania białych znaków.
* Nazewnictwo.
* Strukturę importowania bibliotek.
* Komentarze i dokumentację kodu.

### Rola wcięć w kodzie Python

Języki programowania stosują metodę dzielenia kodu na bloki, w obrębie których istnieją zmienne lub realizowane są funkcje. Najczęściej do tego celu stosuje się nawiasy klamrowe, które jawnie wskazują które instrukcje zawarte są w danym bloku. W przypadku języka Python, do tego celu służą wcięcia. Są one tworzone poprzez 4-krotny znak spacji. Wówczas mówimy o pierwszy poziomie wcięcia. Może zdarzyć się, że wewnątrz pierwszego poziomu, zagnieżdżone są bloki kodu o drugim lub większym poziomie wcięcia i jest to jak najbardziej możliwe. Należy jednak pamiętać o konsekwencji, ponieważ wiele błędów w kodzie, może wynikać właśnie ze złego poziomu wcięcia lub zastosowania złej ilości spacji.

Przykład wielokrotnego poziomu wcięć

|  |
| --- |
| [module\_1\_lesson\_2\_indentation.py] |

### Przykład napisania kodu

W edytorze, utwórzmy zmienną o nazwie *alfabet* i niech to będzie lista, która zawiera litery *a, b, c,* i *d.* Następnie użyjmy konstrukcji *if \_\_name\_\_ == „\_\_main\_\_”:*  i w niej uzyjmy funkcji *print(alfabet)* do wyświetlenia zmiennej. Całość powinna wyglądać tak jak na przykładzie poniżej.

Zadeklarowanie zmiennej alfabet i wyświetlenie jej zawartości

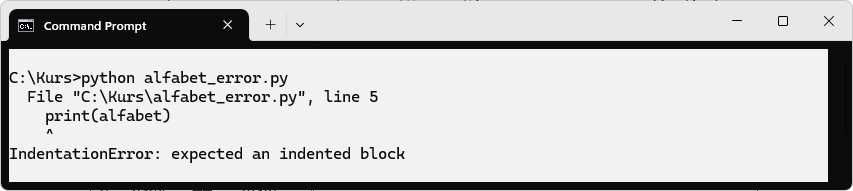
|  |
| --- |
| [module\_1\_lesson\_2\_alfabet.py] |

### Analiza działania kodu i zwracanych błędów

Przeanalizujmy zatem co się po kolei dzieje w kodzie. Na początku tworzona jest zmienna *alfabet*. Jest ona tak zwaną zmienną globalną, ponieważ wartości do niej przypisywane są na samym początku kodu, poza innymi blokami. Następnie tworzony jest pierwszy poziom wcięcia, w którym wyświetlana jest zawartość zmiennej. Jednak co by się stało, gdyby tego wcięcia nie było?

Błedny kod nie zawierający wcięcia

|  |
| --- |
| alfabet = ["a", "b", "c", "d"]  if \_\_name\_\_ == "\_\_main\_\_":  print(alfabet) |



Informacja o typie błędu w terminalu

Program w terminalu wyświetli błąd, ponieważ po dwukropku oczekuje się wcięcia, którego nie ma. Próbując przenosić funkcję *print()* powyżej wyrażenia *if* również zostanie wyświetlony błąd, ponieważ po dwukropku, na końcu tego wyrażenia oczekiwane jest wcięcie i blok kodu.

### Zadania

**Zadanie 1.6.1. Błąd logiczny w Thonny**

* Uruchom Thonny i napisz program, który zawiera błąd logiczny: print(5 / 0)
* Jaki błąd wyświetli się w terminalu?

**Zadanie 1.6.2. Popraw błędy w wcięciach**

|  |
| --- |
| for i in range(3):  print("Pętla zewnętrzna:", i)  for j in range(2):  print(" Pętla wewnętrzna:", j) |

**Zadanie 1.6.3. Popraw formatowanie kodu do PEP 8**

|  |
| --- |
| def oblicz\_pole( dlugosc,szerokosc):  wynik=dlugosc \*szerokosc  return wynik  print (oblicz\_pole(5, 10)) |

**Zadanie 1.6.4. \* Dodanie nowej funkcji**

Zmodyfikuj poniższy kod tak, aby lista alfabet była tworzona wewnątrz funkcji *stworz\_alfabet()*, a następnie wywołaj tę funkcję w bloku *if name == "main"*.

|  |
| --- |
| def stworz\_alfabet():  # Tutaj stwórz listę alfabet if **name** == "**main**": print(alfabet) |

**Zadanie 1.6.5. \* Dodanie nowych elementów do listy**

Zmodyfikuj kod tak, aby program dynamicznie dodawał nowe litery do listy *alfabet*, a następnie wyświetlał jej zawartość Możesz użyć metody *append()*.

|  |
| --- |
| alfabet = ["a", "b", "c", "d"] if **name** == "**main**":  # Dodaj nowe litery do listy print(alfabet) |

## Deklarowanie zmiennych i rola komentarzy

Chociaż deklarowanie zmiennych może wydawać się proste, do tego celu można wykorzystać wiele sposobów. Dobrą praktyką jest to by ich nazwy jasno wskazywały jaką informację przechowują. Jednak czasami jest to nie możliwe, dlatego do opisu zmienny, ale także i kodu, mogą służyć komentarze.

### Deklarowanie zmiennych w Pythonie

Jak już wiemy deklarowanie zmiennych odbywa się poprzez nazwanie jej i przypisanie wartości. Można także to zrobić dla wielu zmiennych, gdy wartość dla nich ma być taka sama. Wówczas pomiędzy zmiennymi należy dać znaki równości, a za ostatnią, wartość jaką mają one mieć.

Wielokrotna deklaracja zmiennych tego samego typu

|  |
| --- |
| [module\_1\_lesson\_3\_multiple\_declarations.py] |

W podobny sposób można do wielu zmiennych przypisywać różne wartości. Wówczas nazwy zmiennych oddzielone są przecinkami, a po znaku przypisania, wartości im odpowiadające również oddzielone przecinkami.

Wielokrotna deklaracja różnych zmiennych

|  |
| --- |
| [module\_1\_lesson\_3\_multiple\_assign.py] |

W pierwszym z wymienionych metod, należy pamiętać, że typ i wartość zmiennych musi być ten sam aby operacja była prawidłowa. Drugi z przykładów pozwala na przypisanie do zmiennych wartości o różnych typach. Programując w języku Python przyjęto pewne zasady nazewnictwa zmiennych których powinno się trzymać:

* Nazwy zmiennych mogą zawierać litery, cyfry i podkreślniki (\_), ale nie mogą zaczynać się od cyfry.
* Zalecany styl to snake\_case czyli używanie podkreślnika gdy nazwa zmiennej składa się z kilku członów (np. moja\_zmienna).
* Dobrą praktyką jest deklarowanie jednej zmiennej w jednej linii aby zachować czytelność kodu.
* Zmienne będące stałymi zapisuje się wielkimi literami

|  |  |
| --- | --- |
|  | Nazwy zmiennych nie mogą zaczynać się i zawierać słów kluczowych takich jak: *if, for, def, class* itp. |

Metody nazywania zmiennych

|  |
| --- |
| [module\_1\_lesson\_3\_variable\_naming.py] |

### Wartość zmiennej i jej typ

Język Python to język dynamicznie typowany. Oznacza to, że typ zmiennej jest przypisywany w momencie jej deklaracji, bez konieczności jawnego określania. Jednocześnie zmienna w trakcie działania programu może zmieniać zarówno swój typ, jak i wartość. Należy jednak pamiętać o zachowaniu kontroli nad tymi zmianami, gdyż w rozbudowanych projektach zmienna zmieniająca typ "w locie" może spowodować błąd działania programu. Omawiany język jest równocześnie na tyle elastyczny, że nowe typy zmiennych mogą się pojawiać poprzez używanie zewnętrznych bibliotek (np. biblioteka *numpy* i jej indywidualny typ *arrays* lub *uint8*).

Nadpisanie typu i wartości

|  |
| --- |
| [module\_1\_lesson\_3\_changing\_type.py] |

Od wersji Pythona 3.5 możliwe jest typowanie zmiennych z adnotacjami i wskazanie spodziewanego typu. Realizuje się to poprzez dopisanie za nazwą zmiennej dwukropka i wskazanie typu (np. *int, str* lub *float*). Nie jest to jednak wymuszone i interpreter nie powstrzyma zapisu wartości typu innego niż spodziewany, natomiast wyświetli odpowiednie ostrzeżenie w terminalu.

Typowanie spodziewanej wartości

|  |
| --- |
| [module\_1\_lesson\_3\_expected\_type.py] |

### Komentarze w kodzie

W strukturze pisanego kodu, można zamieszczać tekst, który nie jest brany pod uwagę przez interpreter. Są to komentarze, których ciąg rozpoczyna się znakiem *#*. Jest to informacja dla interpretera, że ciąg który po nim nastąpi nie ma być brany pod uwagę. Znak kratki dodaje komentarz w linii, w której występuje. Dostępne jest także tworzenie komentarzy wielolinijkowych poprzez użycie znaków *‘’’* lub *”””*. Obydwie formy służą dokumentowania i wyjaśniania co dany fragment kodu robi. Ważne jest aby za jego pomocą nie opisywać danej operacji np. *zwiększenie o jeden* ale dlaczego to robimy np. *niwelowanie błędu zaokrąglenia.*

Używanie komentarzy w kodzie

|  |
| --- |
| [module\_1\_lesson\_3\_comments.py] |

### Zadania

**Zadanie 1.7.1. Deklaracja zmiennej typu całkowitego**

Zadeklaruj zmienną typu całkowitego stosując adnotacje *int*,

**Zadanie 1.7.2. Zmiena typu z int na str**

Zmień typ zmiennej z int na str i wypisz jej nowy typ za pomocą *typ()*.

**Zadanie 1.7.3. Dekalracja liczy zmiennoprzecinkowej**

Zadeklaruj liczbę zmiennoprzecinkową z adnotacją typu float.

**Zadanie 1.7.4. Komentrzenie pojawiaja sie w kodzie**

Wykorzystaj kolejno: komentarz jednoliniowy, wielonijkikowy, oraz komentarz w lini po print("To nie jest komentarz.").

**Zadanie 1.7.5. Inne typy z bibliotek (numpy)**

Użyj typu danych z zewnętrznej biblioteki, np. Numpy. Wypisz wartość np.uint8(255) oraz typ.

## Logika warunkowa

W niniejszym rozdziale zagłębimy się w logikę warunkową, która stanowi fundament podejmowania decyzji przez programy. Logika warunkowa umożliwia dynamiczne sterowanie przepływem wykonywania kodu w zależności od określonych warunków. W Pythonie kluczowe są konstrukcje takie jak if, elif oraz else, które pozwalają na wykonywanie różnych bloków kodu w odpowiedzi na wynik operacji logicznych.

### Wprowadzenie do logiki warunkowej

Logika warunkowa pozwala na podejmowanie decyzji w kodzie na podstawie określonych warunków, wykorzystując instrukcje *if*, *elif* i *else*, a jej działanie opiera się na operatorach logicznych i relacyjnych. Jest to podstawowy element niemal każdego programu, ponieważ umożliwia dynamiczne sterowanie przepływem wykonywania kodu w zależności od zmieniających się danych i warunków.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Przykład** | **Wynik** |
| < | 3 < 7 | Prawda |
| =< | 3 <= 3 | Prawda |
| > | 7 > 3 | Prawda |
| >= | 3 >= 7 | Fałsz |
| == | 3 == 3 | Prawda |
| != | 3 != 7 | Prawda |

Operatory relacyjne (<, <=, >, >=, ==, !=) służą do porównywania wartości i zwracają wynik w postaci *True* lub *False*, co pozwala na budowanie warunków logicznych. Operatory logiczne (*and*, *or*, *not*) umożliwiają łączenie wielu warunków, rozszerzając możliwości sterowania przepływem programu.

Użycie różnych typów operatorów

|  |
| --- |
| [module\_1\_lesson\_3\_operators.py] |

### Instrukcja *if* oraz budowa złożonych wyrażeń

Instrukcja *if* pozwala na wykonywanie określonych bloków kodu w zależności od spełnienia warunków logicznych, umożliwiając sterowanie przepływem programu. Budowa złożonych wyrażeń przy użyciu operatorów logicznych (and, or, not) oraz relacyjnych (<, >, ==, !=, itp.) pozwala na definiowanie bardziej skomplikowanych warunków decyzyjnych.

Kod wewnątrz instrukcji if musi być wcięty o tę samą liczbę spacji, zazwyczaj cztery. Brak lub niespójne wcięcia spowodują błąd składni.

Budowanie wyrażenia if

|  |  |
| --- | --- |
| [module\_1\_lesson\_3\_if.py] |  |

Złożone wyrażenia logiczne w łączą operatory relacyjne z operatorami logicznymi, co pozwala na tworzenie bardziej skomplikowanych warunków decyzyjnych. Dzięki temu program może sprawdzać jednocześnie wiele kryteriów, np. czy liczba jest parzysta i należy do określonego przedziału, bez konieczności stosowania wielu instrukcji.

Złożone wyrażenie warunkowe

|  |
| --- |
| [module\_1\_lesson\_3\_advanced\_if.py] |

### Zadania

**Zadanie 1.8.1. Porównanie dwóch liczb**

* Napisz program, który porówna dwie liczby a i b, a następnie wypisze wynik porównania (*True* lub *False*) dla operatorów <, <=, >, >=, ==, !=.

**Zadanie 1.8.2. Sprawdzenie parzystości**

* Napisz kod, który sprawdzi, czy podana liczba *n* jest parzysta, i zwróci odpowiednią wartość logiczną (*True* dla parzystej, *False* dla nieparzystej).

**Zadanie 1.8.3. Sprawdzenie przedziału**

* Napisz program, który sprawdzi, czy liczba n znajduje się w zakresie od 10 do 50 (włącznie) i zwróci *True* lub *False* bez użycia *if*.

**Zadanie 1.8.4. Sprawdzanie obecności w liście**

* Sprawdź, czy wartość a nie znajduje się w liście [1, 2, 3]. Jeśli a nie należy do tej listy, warunek if powinien to wykryć.

**Zadanie 1.8.5. Liczba nieparzysta, mniejsza niż 50 i podzielna przez 3**

* Wczytaj liczbę b. Chcemy sprawdzić, czy jest:
  + Mniejsza niż 50.
  + Nieparzysta.
  + Dodatkowo musi być podzielna przez 3.
* Stwórz warunek, który spełni te założenia.

## Logika warunkowa w praktyce

Logika warunkowa w programowaniu pozwala na dynamiczne podejmowanie decyzji w oparciu o złożone warunki, m.in. w analizie danych, sztucznej inteligencji i systemach sterowania. Zaawansowane zastosowania obejmują m.in. zagnieżdżone instrukcje *if*, wykorzystanie krótkiej wersji wyrażeń warunkowych (*ternary operator*) oraz operacje na strukturach danych, takich jak listy czy słowniki.

Praktyczne wykorzystanie instrukcji warunkowej

|  |
| --- |
| [module\_1\_lesson\_4\_very\_advanced\_if.py] |

### Zadania

**Zadanie 1.9.1. Analiza przedziałów liczbowych za pomocą *in***

* Sprawdź, czy liczba x znajduje się w przedziale [10, 50], ale nie może być równa 25.

**Zadanie 1.9.2. Automatyczne przypisanie kategorii wiekowej**

* Na podstawie wieku użytkownika wypisz:
  + "Dziecko" – poniżej 13 lat.
  + "Nastolatek" – od 13 do 17 lat.
  + "Dorosły" – od 18 lat wzwyż.

**Zadanie 1.9.3. Sprawdzenie liczby całkowitej**

* Pobierz liczbę od użytkownika i sprawdź:
  + Czy jest liczbą całkowitą (użyj *liczba.isdigit()*).
  + Wypisz jest dodatnia, ujemna czy równa zero.

**Zadanie 1.9.4. \* Sprawdzenie warunków logicznych w liście**

* Wczytaj listę liczb (np 1 2 3 4). Napisz warunek, który sprawdzi, czy lista zawiera przynajmniej jedną liczbę parzystą i żadna z liczb nie jest ujemna (zastosuj *any()* i *all()*).

**Zadanie 1.9.5. \* Walidacja danych użytkownika**

* Napisz warunek *if*, który sprawdzi, czy podane imię użytkownika zawiera tylko litery i ma długość co najmniej 3 znaki.

## Konwersje liczb

W języku Python dostępne są różne typy danych, takie jak liczby całkowite (*int*), liczby zmiennoprzecinkowe (*float*), ciągi znaków (*str*) oraz wartości logiczne (*bool*). Każdy z nich ma swoje zastosowanie – *int* służy do reprezentacji liczb całkowitych, *float* do liczb zmiennoprzecinkowych, *str* do przechowywania tekstu, a *bool* do wartości logicznych *True* lub *False*.

Konwersja umożliwia zmianę typu danych, np. int("64") zamienia tekst na liczbę, a str(3.14) liczbę na tekst. bool(0) zwraca False, a bool(1) True. Python automatycznie rzutuje liczby całkowite na float w wyrażeniach mieszanych.

Wartości 0, "" i None traktowane są jako False, a pozostałe jako True. Dzięki temu konwersja jest przydatna w instrukcjach warunkowych i pętlach, np. if liczba: sprawdza, czy wartość jest różna od zera. Do rzutowania służą specjalne funkcje wbudowane i tak:

* Za pomocą funkcji *int()* można konwertować typy danych na typ całkowity
* Za pomocą funkcji *float()* można konwertować typy danych na typ zmiennoprzecinkowy
* Za pomocą funkcji *complex()* można konwertować typy danych na typ liczby zespolonej
* Za pomocą funkcji *bool()* można konwertować typy danych na typ bool’owski
* Za pomocą funkcji *str()* można konwertować typy danych na łańcuchy znaków
* Za pomocą funkcji *list()* można konwertować typy danych na tablicę
* Za pomocą funkcji *tuple()* można konwertować typy danych na krotkę
* Za pomocą funkcji *set()* można konwertować typy danych na zbiór
* Za pomocą funkcji *dict()* można konwertować typy danych na słownik

Zastosowanie operacji konwersji typów

|  |
| --- |
| [module\_1\_lesson\_4\_conversion.py] |

### Zadania

**Zadanie 1.10.1. Konwersja stringa na liczbę**

* Pobierz od użytkownika liczbę w postaci stringa i skonwertuj ją na int oraz float, następnie wyświetl oba wyniki.

**Zadanie 1.10.2. Zamiana listy na krotkę**

* Pobierz od użytkownika ciąg liczb oddzielonych spacją, skonwertuj je na listę, a następnie zamień listę na krotkę i wyświetl wynik.

**Zadanie 1.10.3. Niejawna konwersja typów**

* Pobierz dwie liczby – jedną całkowitą, drugą zmiennoprzecinkową – dodaj je do siebie i wypisz wynik oraz jego typ.

**Zadanie 1.10.4. \* Konwersja listy znaków na string**

* Pobierz od użytkownika słowo, przekształć je na listę znaków, a następnie zamień tę listę z powrotem na string użyj (.*any(*)).

**Zadanie 1.10.5. \* Konwersja zagnieżdżona i operacje na typach danych**

* Pobierz od użytkownika ciąg liczb oddzielonych przecinkami, zamień go na listę liczb całkowitych, oblicz ich sumę oraz średnią i wypisz wyniki wraz z odpowiednimi typami danych.

## Działania matematyczne

Python obsługuje różne rodzaje liczb, a najczęściej używane to liczby całkowite (*int*) i zmiennoprzecinkowe (*float*). Dodatkowo, dostępne są liczne funkcje i operatory matematyczne, które umożliwiają wykonywanie obliczeń i manipulowanie wartościami liczbowymi.

Zastosowanie wbudowanych operacji matematycznych

|  |
| --- |
| [module\_1\_lesson\_4\_numbers.py] |

### Zadania

**Zadanie 1.11.1. Typy liczb**

* Wczytaj liczbę całkowitą oraz zmiennoprzecinkową i wypisz ich typy

**Zadanie 1.11.2. Zaokrąglanie**

* Zaokrąglij liczbę do 3 miejsc

**Zadanie 1.11.3. Formatowanie liczby w komunikacie**

* Wczytaj liczbę i wyświetl ją w zdaniu z dokładnością do 2 miejsc po przecinku.

**Zadanie 1.11.4. Operacje matematyczne**

1. Wczytaj dwie liczby *float* i wykonaj na nich podstawowe operacje matematyczne.

**Zadanie 1.11.5. \* Formatowanie procentów**

Wczytaj liczbę i wyświetl ją w formacie procentowym.

## Podsumowanie

* Python posiada czytelną składnię, ma podstawy proste do przyswojenia i jest bardzo wszechstronny. Jednocześnie ma interpretowany charakter oraz potrafi mieć problemy z wydajnością pisanego w nim kodu
* Cechą języka jest, że można z niego korzystać w formie interaktywnej konsoli czyli REPL – Read-Eval-Print-Loop, ale również pozwala tworzyć pliki i uruchamiać instrukcje w formie skryptowej.
* Złożone zadania i problemy warto dzielić na zestaw mniejszych kroków i realizować je w formie algorytmu. Pozwoli to na znalezienie najbardziej optymalnego rozwiązania minimalizując złożoność obliczeniową i pamięciową
* Aby zainstalować interpreter języka Python, na komputerze z systemem Windows, możesz skorzystać z oficjalnej strony projektu pobierając instalator. Z kolei na systemie Linux, możesz skorzystać z komendy *apt install* w terminalu i wpisać wersję Pythona jaka cię interesuje.
* Z języka Python można korzystać w trybie interaktywnym w terminalu lub jako sekwencja instrukcji w formie skryptu pliku z rozszerzeniem *.py*
* Python oferuje wiele rodzajów zmiennych, których typ określa automatycznie.
* Od użytkownika możliwe jest pobranie danych w czasie działania programu, za pomocą funkcji *input()* jak również ich wyświetlenie za pomocą funkcji *print().*
* IDE to zintegrowane środowisko programistyczne, które ma za zadanie zapewnić wszelkie niezbędne narzędzia i komfort w pisaniu oprogramowania w języku Python
* Dostępnych jest wiele różnych IDE które mają swoje wady i zalety. Od użytkownika zależy który z nich spełni jego wymagania, dlatego warto się z nim zaznajomić
* Python nie używa standardowych klamr do dzielenia kodu na bloki. W zamian teog wykorzystuje wcięcia
* PEP to zbiór dokumentów dotyczących technologii programowania w języku Python. Jeden z nich to PEP 8 mówiący o stylu pisania kodu.
* Zmienne mogą być deklarowane zbiorczo w jednej linii, ale zaleca się każda z nich deklarować osobno, dla przejrzystości kodu.
* W wieloczłonowych nazwach zmiennych, wykorzystuje się zamiast spacji znak podkreślnika (\_), który oddziela wyrazy od siebie.
* W nazwach zmiennych nie wolno wykorzystywać słów kluczowych jak *if, for* czy *def*
* Rolą komentarzy w kodzie jest tłumaczenie dlaczego dana operacja jest wykonywana. Za pomocą kratki (*#*) tworzy się komentarz jednolinijkowy.

## Zadania podsumowujące moduł

**Zadanie 1.13.1. Instalacja danej wersji Pythona**

* Zainstaluj Python w wersji 3.11, bez usuwania poprzedniej wersji.

**Zadanie 1.13.2. Tworzenie skryptu i wyświetlanie wartości w terminalu**

* Napisz skrypt, który wypisuje na ekranie komunikat „Hello, World!” za pomocą funkcji print().

**Zadanie 1.13.3. Znaczenie komentrzy**

* Wyjaśnij znaczenie wcięć w języku Python.

**Zadanie 1.13.4. Zasady PEP 8**

* Co oznacza skrót PEP i dlaczego PEP 8 jest ważny?

**Zadanie 1.13.5. Sprawdzenie poprawności liczby w podanym zakresie**

* Pobierz liczbę od użytkownika i sprawdź, czy mieści się w zakresie 1-100. Jeśli nie, poproś o drugie wprowadzenie.

**Zadanie 1.13.6. Weryfikacja numeru telefonu**

* Pobierz numer telefonu w formacie xxx-xxx-xxx, rozdziel łańcuch za pomocą ‘-’ i sprawdź, czy każdy powstały zbiór posiada 3 cyfry.

**Zadanie 1.13.7. Dzielenie całkowite**

* Wczytaj dwie liczby i oblicz wynik dzielenia całkowitego (//) i sprawdź typ wyniku.

**Zadanie 1.13.8. Obliczanie kwadratu i pierwiastka**

* Wczytaj liczbę, oblicz jej kwadrat oraz pierwiastek kwadratowy (za pomocą \*\* oraz *math.sqrt()*).

**Zadanie 1.13.9. \* Obliczanie pola podstawy i objętości walca**

Napisz skrypt, w którym

* Użytkownik poda promień okręgu i wysokość walca
* Program wyświetli pole podstawy walca oraz jego objętość

**Zadanie 1.13.10. \* Obliczanie średniej z ocen**

Napisz program, który

* Poprosi użytkownika o podanie ciągu ocen oddzielonych spacjami
* Przekonwertuje podane oceny na listę liczb całkowitych
* Obliczy i wyświetli średnią ocen

**Zadanie 1.13.11. Obliczanie średniej i dynamiczna zmiana typu zmiennej**

* Oblicz średnią z podanych przez użytkownika 4 liczb. Sumę liczb zapisz w zmiennej *suma\_z\_liczb* a następnie nadpisz ją, licząc z niej średnią. Wyświetl typ tej zmiennej przed i po nadpisaniu.

**Zadanie 1.13.12. Zmienna *float***

* Zadeklaruj zmienną cena jako float, przypisz jej wartość 3.14 i wypisz jej typ

**Zadanie 1.13.13. Automatyczne rozpoznawanie formatu daty**

* Pobierz datę w jednym z formatów (YYYY-MM-DD, DD.MM.YYYY, MM/DD/YYYY), rozpoznaj format i przekonwertuj ją do YYYY-MM-DD.

**Zadanie 1.13.14. Sprawdzenie czy rok jest przestępny**

* Pobierz rok i sprawdź, czy jest przestępny (podzielny przez 4, ale nie przez 100, chyba że przez 400).

**Zadanie 1.13.15 Zliczanie słów w zdaniu**

Napisz program, który

* Pobierze od użytkownika ciąg znaków
* Wyświetli liczbę słów i pominie białe znaki

**Zadanie 1.13.16 Obliczanie BMI**

Napisz program, który

* Pobierze od użytkownika wzrost w centymetrach i wagę w kilogramach
* Obliczy BMI przy użyciu następującego wzoru:

**Zadanie 1.13.17. Zmienna logiczna**

* Zadeklaruj zmienną bool i przekonwertuj na int.

**Zadanie 1.13.18. Dodawanie int i float**

* Utwórz zmienną liczba i przypisz jej int, następnie dodaj do niej float i wypisz wynik.

**Zadanie 1.13.19.** Konwersja jednostek masy

* Pobierz wagę w kilogramach i przelicz ją na gramy oraz funty.

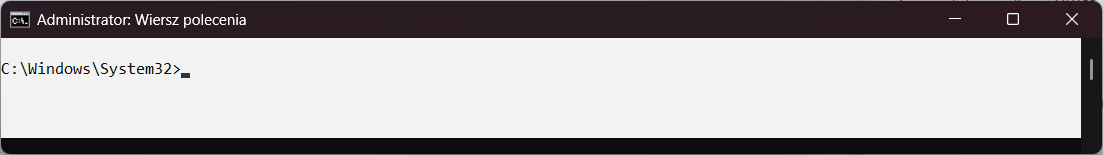
**Zadanie 1.13.20. Sprawdzenie poprawności numeru NIP**

* Pobierz numer NIP od użytkownika i sprawdź, czy składa się dokładnie z 10 cyfr oraz spełnia podstawowe warunki walidacyjne.

## Dodatek

**Zadanie 1.1.1.**

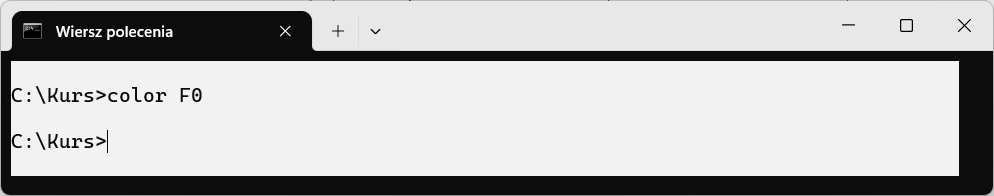
Jedną z metod uruchomienia konsoli z prawami administratora jest wpisanie *CMD* w oknie wyszukiwania systemu Windows, wybranie prawym przyciskiem myszy *Wiersza Poleceń*, a następnie *Uruchom jako administrator*.

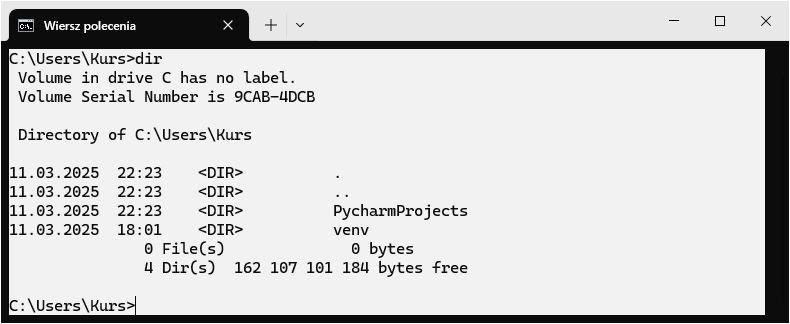


**Zadanie 1.1.2.**

* Uruchom Visual Studio Code
  + Kliknij Start → wpisz Visual Studio Code → naciśnij Enter.
  + Możesz też otworzyć terminal (*Win + R* → wpisz *code* → Enter).
* Zmiana motywu na jasny
  + Otwórz Ustawienia (*Ctrl + Shift + P* → wpisz *Preferences: Color Them*e).
  + Wybierz Light Theme (np. "Light+").
* Potwierdź wybór.

**Zadanie 1.1.3.**

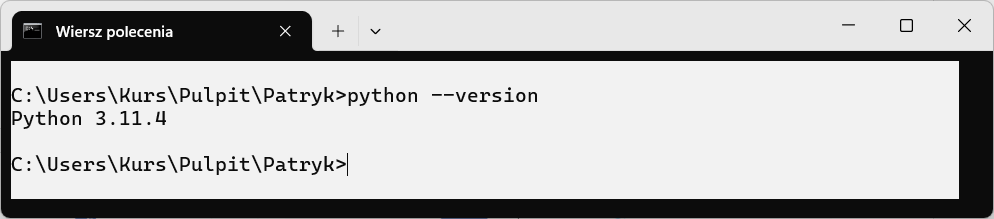


**Zadanie 1.1.4.**

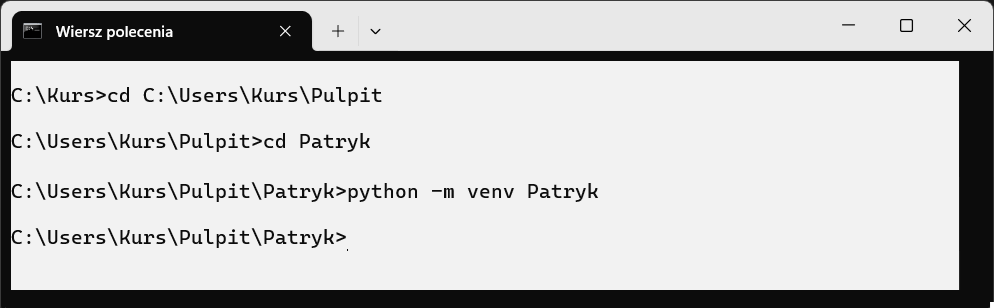
**Zadanie 1.1.5.**

|  |
| --- |
| [module\_1\_lesson\_1\_1\_1\_5.py] |

**Zadanie 1.2.1.**



**Zadanie 1.2.2.**

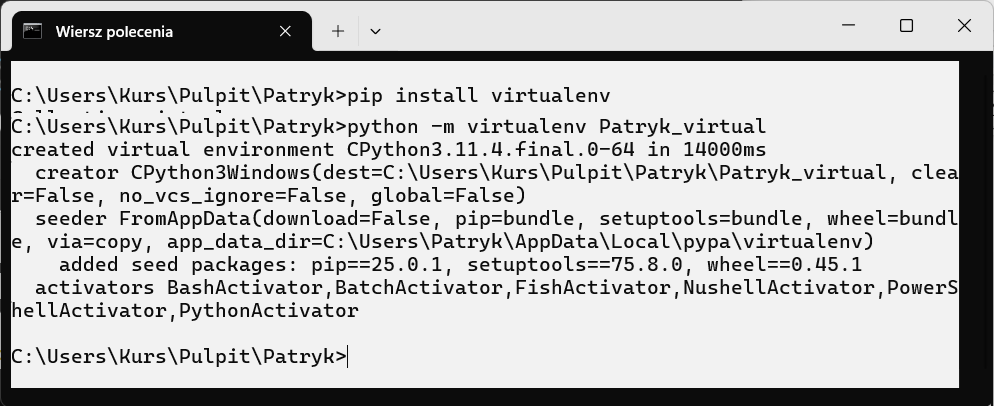


**Zadanie 1.2.3.**

|  |
| --- |
| [module\_1\_lesson\_1\_1\_2\_3.py] |

[module\_1\_lesson\_1\_1\_2\_3.png]

**Zadanie 1.2.4.**

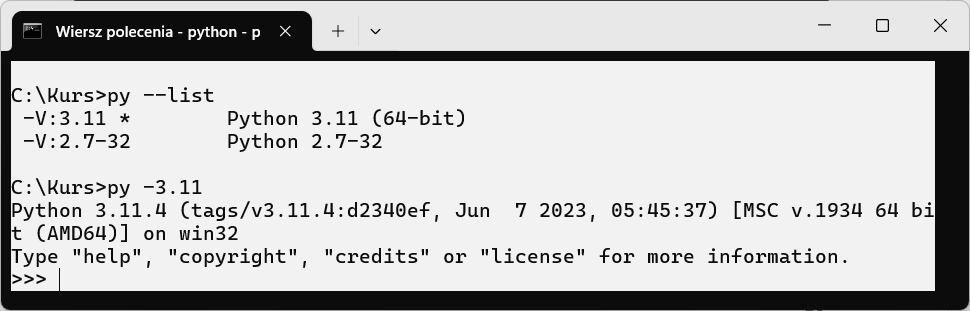


**Zadanie 1.2.5.**

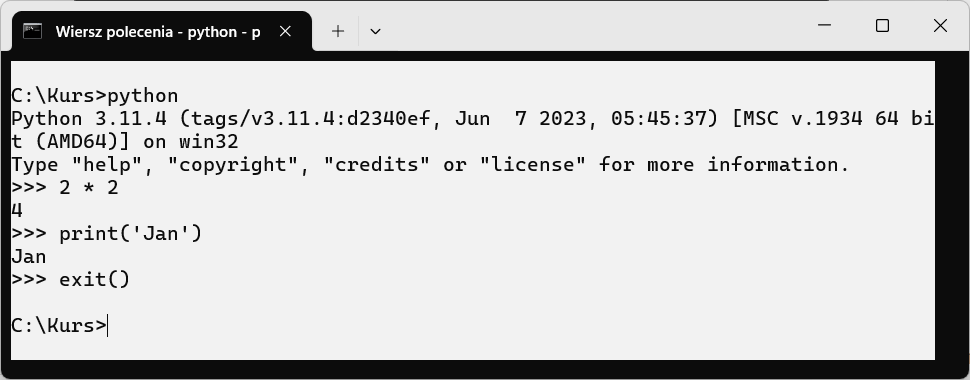
|  |
| --- |
| [module\_1\_lesson\_1\_1\_2\_5.py] |

[module\_1\_lesson\_1\_1\_2\_5.png]

**Zadanie 1.3.1.**



**Zadanie 1.3.2.**



**Zadanie 1.3.3.**

|  |
| --- |
| [module\_1\_lesson\_1\_1\_3\_3.py] |

[module\_1\_lesson\_1\_1\_3\_3.png]

**Zadanie 1.3.4.**

|  |
| --- |
| [module\_1\_lesson\_1\_1\_3\_4.py] |

[module\_1\_lesson\_1\_1\_3\_4.png]

**Zadanie 1.3.5.**

|  |
| --- |
| [module\_1\_lesson\_1\_1\_3\_5.py] |

[module\_1\_lesson\_1\_1\_3\_5.png]

**Zadanie 1.4.1.**

|  |
| --- |
| [module\_1\_lesson\_2\_1\_4\_1.py] |

[module\_1\_lesson\_2\_1\_4\_1.png]

**Zadanie 1.4.2.**

|  |
| --- |
| [module\_1\_lesson\_2\_1\_4\_2.py] |

[module\_1\_lesson\_2\_1\_4\_2.png]

**Zadanie 1.4.3.**

|  |
| --- |
| [module\_1\_lesson\_2\_1\_4\_3.py] |

[module\_1\_lesson\_2\_1\_4\_3.png]

**Zadanie 1.4.4.**

|  |
| --- |
| [module\_1\_lesson\_2\_1\_4\_4.py] |

[module\_1\_lesson\_2\_1\_4\_4.png]

**Zadanie 1.4.5.**

|  |
| --- |
| [module\_1\_lesson\_2\_1\_4\_5.py] |

[module\_1\_lesson\_2\_1\_4\_5.png]

**Zadanie 1.5.1.**

Uruchom IDE, np Thonny, wklej kod: print("Witaj w Pythonie!")  
Natępnie uruchom program.

**Zadanie 1.5.2.**

|  |
| --- |
| [module\_1\_lesson\_2\_1\_5\_2.py] |

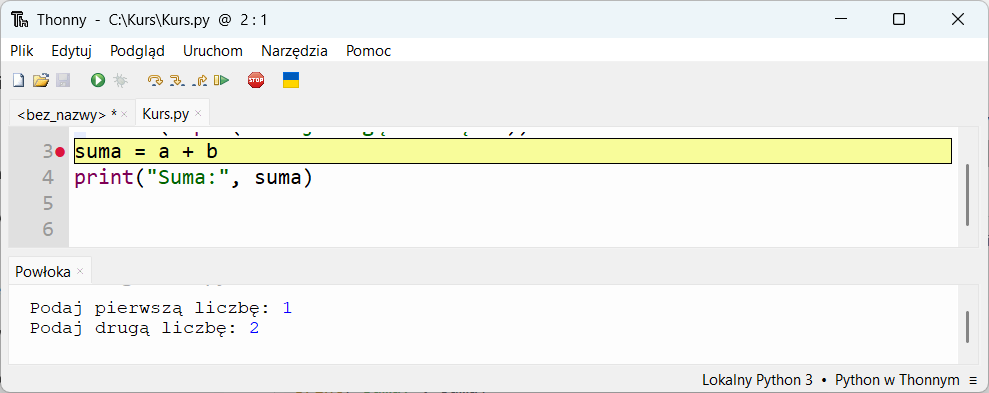
[module\_1\_lesson\_2\_1\_5\_2.png]

**Zadanie 1.5.3.**

|  |
| --- |
| [module\_1\_lesson\_2\_1\_5\_3.py] |

|  |
| --- |
|  |

**Zadanie 1.5.4.**

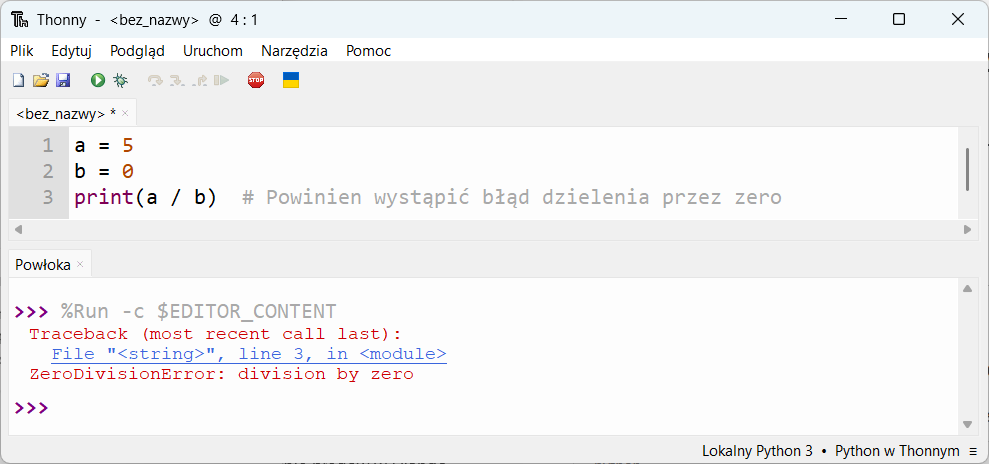


**Zadanie 1.5.5.**

|  |
| --- |
| [module\_1\_lesson\_2\_1\_5\_5.py] |

|  |
| --- |
|  |

**Zadanie 1.6.1.**



**Zadanie 1.6.2.**

|  |
| --- |
| [module\_1\_lesson\_2\_1\_6\_2.py] |

[module\_1\_lesson\_2\_1\_6\_2.png]

**Zadanie 1.6.3.**

|  |
| --- |
| [module\_1\_lesson\_2\_1\_6\_3.py] |

[module\_1\_lesson\_2\_1\_6\_3.png]

**Zadanie 1.6.4.**

|  |
| --- |
| [module\_1\_lesson\_2\_1\_6\_4.py] |

[module\_1\_lesson\_2\_1\_6\_4.png]

**Zadanie 1.6.5.**

|  |
| --- |
| [module\_1\_lesson\_2\_1\_6\_5.py] |

[module\_1\_lesson\_2\_1\_6\_5.png]

**Zadanie 1.7.1.**

|  |
| --- |
| [module\_1\_lesson\_3\_1\_7\_1.py] |

[module\_1\_lesson\_3\_1\_7\_1.png]

**Zadanie 1.7.2.**

|  |
| --- |
| [module\_1\_lesson\_3\_1\_7\_2.py] |

[module\_1\_lesson\_3\_1\_7\_2.png]

**Zadanie 1.7.3.**

|  |
| --- |
| [module\_1\_lesson\_3\_1\_7\_3.py] |

[module\_1\_lesson\_3\_1\_7\_3.png]

**Zadanie 1.7.4.**

|  |
| --- |
| [module\_1\_lesson\_3\_1\_7\_4.py] |

[module\_1\_lesson\_3\_1\_7\_4.png]

**Zadanie 1.7.5.**

|  |
| --- |
| [module\_1\_lesson\_3\_1\_7\_5.py] |

[module\_1\_lesson\_3\_1\_7\_5.png]

**Zadanie 1.8.1.**

|  |
| --- |
| [module\_1\_lesson\_3\_1\_8\_1.py] |

[module\_1\_lesson\_3\_1\_8\_1.png]

**Zadanie 1.8.2.**

|  |
| --- |
| [module\_1\_lesson\_3\_1\_8\_2.py] |

[module\_1\_lesson\_3\_1\_8\_2.png]

**Zadanie 1.8.3.**

|  |
| --- |
| [module\_1\_lesson\_3\_1\_8\_3.py] |

[module\_1\_lesson\_3\_1\_8\_3.png]

**Zadanie 1.8.4.**

|  |
| --- |
| [module\_1\_lesson\_3\_1\_8\_4.py] |

[module\_1\_lesson\_3\_1\_8\_4.png]

**Zadanie 1.8.5.**

|  |
| --- |
| [module\_1\_lesson\_3\_1\_8\_5.py] |

[module\_1\_lesson\_3\_1\_8\_5.png]

**Zadanie 1.9.1.**

|  |
| --- |
| [module\_1\_lesson\_4\_1\_9\_1.py] |

[module\_1\_lesson\_4\_1\_9\_1.png]

**Zadanie 1.9.2.**

|  |
| --- |
| [module\_1\_lesson\_4\_1\_9\_2.py] |

[module\_1\_lesson\_4\_1\_9\_2.png]

**Zadanie 1.9.3.**

|  |
| --- |
| [module\_1\_lesson\_4\_1\_9\_3.py] |

[module\_1\_lesson\_4\_1\_9\_3.png]

**Zadanie 1.9.4.**

|  |
| --- |
| [module\_1\_lesson\_4\_1\_9\_4.py] |

[module\_1\_lesson\_4\_1\_9\_4.png]

**Zadanie 1.9.5.**

|  |
| --- |
| [module\_1\_lesson\_4\_1\_9\_5.py] |

[module\_1\_lesson\_4\_1\_9\_5.png]

**Zadanie 1.10.1.**

|  |
| --- |
| [module\_1\_lesson\_4\_1\_10\_1.py] |

[module\_1\_lesson\_4\_1\_10\_1.png]

**Zadanie 1.10.2.**

|  |
| --- |
| [module\_1\_lesson\_4\_1\_10\_2.py] |

[module\_1\_lesson\_4\_1\_10\_2.png]

**Zadanie 1.10.3.**

|  |
| --- |
| [module\_1\_lesson\_4\_1\_10\_3.py] |

[module\_1\_lesson\_4\_1\_10\_3.png]

**Zadanie 1.10.4.**

|  |
| --- |
| [module\_1\_lesson\_4\_1\_10\_4.py] |

[module\_1\_lesson\_4\_1\_10\_4.png]

**Zadanie 1.10.5.**

|  |
| --- |
| [module\_1\_lesson\_4\_1\_10\_5.py] |

[module\_1\_lesson\_4\_1\_10\_5.png]

**Zadanie 1.11.1.**

|  |
| --- |
| [module\_1\_lesson\_4\_1\_11\_1.py] |

[module\_1\_lesson\_4\_1\_11\_1.png]

**Zadanie 1.11.2.**

|  |
| --- |
| [module\_1\_lesson\_4\_1\_11\_2.py] |

[module\_1\_lesson\_4\_1\_11\_2.png]

**Zadanie 1.11.3.**

|  |
| --- |
| [module\_1\_lesson\_4\_1\_11\_3.py] |

[module\_1\_lesson\_4\_1\_11\_3.png]

**Zadanie 1.11.4.**

|  |
| --- |
| [module\_1\_lesson\_4\_1\_11\_4.py] |

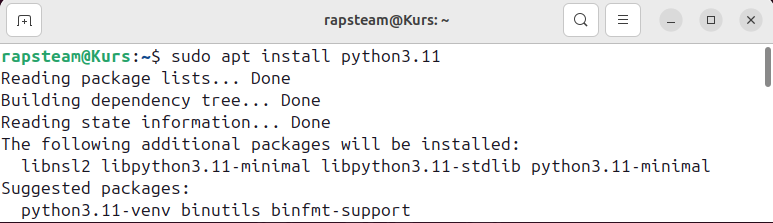
[module\_1\_lesson\_4\_1\_11\_4.png]

**Zadanie 1.11.5.**

|  |
| --- |
| [module\_1\_lesson\_4\_1\_11\_5.py] |

[module\_1\_lesson\_4\_1\_11\_5.png]

**Zadanie** **1.13.1.**



**Zadanie 1.13.2.**

|  |
| --- |
| [module\_1\_summary\_1\_13\_2.py] |

[module\_1\_summary\_1\_13\_2.png]

**Zadanie 1.13.3.**

Wcięcia w Pythonie są nieodłaczne od języka, ponieważ określają strukturę kodu. Python nie używa nawiasów klamrowych {} do wyznaczania bloków kodu, a zamiast tego wymaga wcięć (najczęściej 4 spacje).

**Zadanie 1.13.4.**

PEP oznacza Python Enhancement Proposal, a PEP 8 to dokument określający standardy pisania czytelnego i dobrze sformatowanego kodu w Pythonie. Pomaga programistom utrzymać jednolity styl kodowania.

**Zadanie 1.13.5.**

|  |
| --- |
| [module\_1\_summary\_1\_13\_5.py] |

[module\_1\_summary\_1\_13\_5.png]

**Zadanie 1.13.6.**

|  |
| --- |
| [module\_1\_summary\_1\_13\_6.py] |

[module\_1\_summary\_1\_13\_6.png]

**Zadanie 1.13.7.**

|  |
| --- |
| [module\_1\_summary\_1\_13\_7.py] |

[module\_1\_summary\_1\_13\_7.png]

**Zadanie 1.13.8.**

|  |
| --- |
| [module\_1\_summary\_1\_13\_8.py] |

[module\_1\_summary\_1\_13\_8.png]

**Zadanie 1.13.9.**

|  |
| --- |
| [module\_1\_summary\_1\_13\_9.py] |

[module\_1\_summary\_1\_13\_9.png]

**Zadanie 1.13.10.**

|  |
| --- |
| [module\_1\_summary\_1\_13\_10.py] |

[module\_1\_summary\_1\_13\_10.png]

**Zadanie 1.13.11.**

|  |
| --- |
| [module\_1\_summary\_1\_13\_11.py] |

[module\_1\_summary\_1\_13\_11.png]

**Zadanie 1.13.12.**

|  |
| --- |
| [module\_1\_summary\_1\_13\_12.py] |

[module\_1\_summary\_1\_13\_12.png]

**Zadanie 1.13.13.**

|  |
| --- |
| [module\_1\_summary\_1\_13\_13.py] |

[module\_1\_summary\_1\_13\_13.png]

**Zadanie 1.13.14.**

|  |
| --- |
| [module\_1\_summary\_1\_13\_14.py] |

[module\_1\_summary\_1\_13\_14.png]

**Zadanie 1.13.15.**

|  |
| --- |
| [module\_1\_summary\_1\_13\_15.py] |

[module\_1\_summary\_1\_13\_15.png]

**Zadanie 1.13.16.**

|  |
| --- |
| [module\_1\_summary\_1\_13\_16.py] |

[module\_1\_summary\_1\_13\_16.png]

**Zadanie 1.13.17.**

|  |
| --- |
| [module\_1\_summary\_1\_13\_17.py] |

[module\_1\_summary\_1\_13\_17.png]

**Zadanie 1.13.18.**

|  |
| --- |
| [module\_1\_summary\_1\_13\_18.py] |

[module\_1\_summary\_1\_13\_18.png]

**Zadanie 1.13.19.**

|  |
| --- |
| [module\_1\_summary\_1\_13\_19.py] |

[module\_1\_summary\_1\_13\_19.png]

**Zadanie 1.13.20.**

|  |
| --- |
| [module\_1\_summary\_1\_13\_20.py] |

[module\_1\_summary\_1\_13\_20.png]

# Struktury danych

W niniejszym module skupimy się na strukturach danych w Pythonie – kluczowych narzędziach, które pozwalają przechowywać, organizować i przetwarzać informacje w programach. Omówimy zarówno proste struktury, takie jak listy, krotki i łańcuchy znaków, jak i bardziej złożone, takie jak słowniki i zbiory.

## Zapoznanie z obiektami: Listy, krotki i słowniki

W Pythonie istnieje kilka podstawowych struktur danych, które pozwalają na efektywne przechowywanie i manipulowanie informacjami: listy, krotki oraz słowniki. Struktury te są szeroko wykorzystywane w programowaniu – od przechowywania wyników operacji, przez zarządzanie konfiguracjami, aż po modelowanie baz danych czy obsługę API

**Listy**

Listy to dynamiczne, uporządkowane kolekcje elementów, które można modyfikować – dodawać (append()), usuwać (remove(), pop()) i sortować (sort()).

|  |
| --- |
| [module\_2\_lesson\_1\_lists.py] |

**Krotki**

Krotki przypominają listy, ale są niemutowalne, co oznacza, że ich zawartość nie może być zmieniona po utworzeniu; sprawdzają się, gdy dane powinny pozostać niezmienne, np. współrzędne punktów czy konfiguracje.

|  |
| --- |
| [module\_2\_lesson\_1\_tuples.py] |

**Słowniki**

Słowniki przechowują dane w formie par klucz-wartość, co umożliwia szybki dostęp do informacji i łatwe ich modyfikowanie.

|  |
| --- |
| [module\_2\_lesson\_1\_dicts.py] |

### Zadania

**Zadanie 2.1.1. Lista owoców**

* Wczytaj trzy nazwy owoców, dodaj je do listy i wyświetl całą listę.

**Zadanie 2.1.2. Modyfikacja listy liczb**

* Utwórz listę pięciu liczb, usuń ostatni element, dodaj nową liczbę i wyświetl wynik.

**Zadanie 2.1.3. Krotka z danymi użytkownika**

* Wczytaj imię, wiek i miasto, a następnie zapisz je w krotce i wyświetl.

**Zadanie 2.1.4.** **Słownik z danymi samochodu**

* Utwórz słownik przechowujący markę, model i rok produkcji samochodu, a następnie wyświetl jego zawartość.

**Zadanie 2.1.5. \*** Iteracja po słowniku

* Wczytaj trzy przedmioty szkolne i ich oceny, zapisz je w słowniku, a następnie wyświetl każdą parę klucz-wartość.

## Tworzenie napisów i działanie na łańcuchach znaków

Napisy (ang. strings) to jedne z najczęściej używanych typów danych w programowaniu. W języku Python są one reprezentowane jako sekwencje znaków ujęte w pojedyncze ('tekst') lub podwójne ("tekst") cudzysłowy. Można je traktować jako uporządkowane ciągi znaków, w których każdy element ma swój indeks – od 0 dla pierwszego znaku do *n-1*, gdzie n to długość napisu.

|  |
| --- |
| [module\_2\_lesson\_2\_strings.py] |

W interaktywnych programach często zachodzi potrzeba pobrania danych od użytkownika. Służy do tego funkcja *input()*, która zawsze zwraca wynik jako napis (string), nawet jeśli użytkownik wpisze liczbę. Z kolei na tych stringach można wykonywać manipulacje za pomocą gotowych funkcji lub przekonwertować dane wejściowe na liczby.

|  |
| --- |
| [module\_2\_lesson\_2\_strings\_operations.py] |

Często chcemy wstawić wartości zmiennych do napisu w czytelny sposób. Możemy to zrobić na kilka sposobów, za pomocą *f-strings*, *format()*, stosując *%*. Metoda *f-strings* jest obecnie najczęściej używana, ponieważ są bardziej czytelne i wydajne.

|  |
| --- |
| [module\_2\_lesson\_2\_strings\_formatting.py] |

### Zadania

**Zadanie 2.2.1. Imię z wielkiej litery**

* Wczytaj imię, sprawdź, czy zaczyna się z wielkiej litery, jeśli nie – popraw.

**Zadanie 2.2.2. Łączenie danych za pomocą f-string**

* Wczytaj imię, wiek i miasto, a następnie wyświetl je w zdaniu.

**Zadanie 2.2.3. Formatowanie długości napisu**

* Wczytaj dowolny tekst i wyświetl jego długość w zdaniu.

**Zadanie 2.2.4. Wyświetlanie danych w kolumnach**

* Wczytaj tekst i zamień wszystkie litery na wielkie, a potem na małe.

**Zadanie 2.2.5. \* Formatowanie procentów**

* Wczytaj liczbę i wyświetl ją w formacie procentowym.

## Podsumowanie

* Trzy główne typy obiektów w Pythonie: listy, krotki i słowniki, służą do przechowywania i manipulowania informacjami.
* Listy to dynamiczne, uporządkowane i modyfikowalne kolekcje, które umożliwiają dodawanie, usuwanie i sortowanie elementów.
* Krotki są podobne do list, ale są niemutowalne, co sprawia, że nadają się do przechowywania danych, które nie powinny ulec zmianie.
* Słowniki przechowują dane w postaci par klucz-wartość, umożliwiając szybki dostęp do informacji i ich modyfikację.
* Napisy (stringi) w Pythonie są reprezentowane jako sekwencje znaków ujęte w pojedyncze, podwójne lub potrójne cudzysłowy.
* Łańcuchy znaków można łączyć, powielać i zmieniać wielkość liter
* Do wyświetlania sformatowanych napisów służą f-stringi z metodą format(). Do tego można używać operatora %

## Zadania podsumowujące moduł

**Zadanie 2.4.1. Zamiana liter**

* Wczytaj tekst od użytkownika i zamień wszystkie litery „a” na „@”.

**Zadanie 2.4.2. Sprawdzanie końcówki**

* Wczytaj tekst i sprawdź, czy kończy się kropką. Jeśli nie – dodaj kropkę.

**Zadanie 2.4.3. Liczenie znaków**

* Wczytaj tekst i policz, ile razy występuje w nim litera „a”.

**Zadanie 2.4.4. Skracanie tekstu**

* Wczytaj tekst i skróć go do pierwszych 10 znaków.

**Zadanie 2.4.5. Odwracanie tekstu**

* Wczytaj tekst i wyświetl go w odwrotnej kolejności.

**Zadanie 2.4.6. Obliczanie wartości bezwzględnej i potęgowania**

* Wczytaj liczbę (ujemną) i oblicz jej wartość bezwzględną za pomocą funkcji *abs(*). Następnie podnieś tę liczbę do potęgi trzeciej za pomocą funkcji *pow()*.

**Zadanie 2.4.7. Dzielenie całkowite**

* Wczytaj dwie liczby i oblicz wynik dzielenia całkowitego (//) i sprawdź typ wyniku.

**Zadanie 2.4.8. Obliczanie kwadratu i pierwiastka**

* Wczytaj liczbę, oblicz jej kwadrat oraz pierwiastek kwadratowy (za pomocą \*\* oraz *math.sqrt()*).

**Zadanie 2.4.9.** **Liczba do potęgi**

* Wczytaj liczbę i podnieś ją do potęgi drugiej.

**Zadanie 2.4.10. Obliczanie średniej z użyciem list**

* Wczytaj trzy liczby i zapisz je do listy za pomocą metody *append()*. Następnie użyj funkcji *len()* i podziel ich sumę przez zwróconą przez sumę wartość. Wyświetl wynik.

**Zadanie 2.4.11.** **Dodawanie i usuwanie elementów w liście**

* Utwórz listę *garderoba[]*. Dodaj do niej koszulę, buty, bieliznę, spodnie i kubek. Wyświetl jej zawartość. Następnie usuń ostatni element listy za pomocą metody *pop()* i nie pasujący element za pomocą metody *remove().*Wyświetl zawartość po zmianach.

**Zadanie 2.4.12. Znajdowanie największej liczby**

* Wczytaj pięć liczb do listy i znajdź największą wartość.

**Zadanie 2.4.13. Usuwanie elementu**

* Wczytaj listę pięciu liczb i usuń pierwszy element.

**Zadanie 2.4.14. Sortowanie listy**

* Wczytaj pięć liczb i posortuj je w kolejności rosnącej.

**Zadanie 2.4.15. Suma elementów listy**

* Wczytaj pięć liczb i oblicz ich sumę.

**Zadanie 2.4.16. Rozpakowanie krotki**

* Stwórz krotkę *punkt* o współrzędnych 7, 12, 4. Następnie rozpakuj ją przypisując jej elementy do postępujących po sobie zmiennych *x, y, z*. Wypisz te zmienne.

**Zadanie 2.4.17. Odczyt z krotki**

* Stwórz krotkę z trzema wartościami, a następnie wyświetl drugi element.

**Zadanie 2.4.18. Dodawanie danych do słownika**

* Wczytaj nazwę kraju i jego stolicę, a następnie zapisz te dane w słowniku.

**Zadanie 2.4.19. Modyfikacja wartości w słowniku**

* Stwórz słownik przechowujący imię i wiek. Następnie zwiększ wiek o 1.

**Zadanie 2.4.20. \* Modyfikowanie zagnieżdżonych struktur**

* Stwórz krotkę zawierającą imię, wiek ucznia i przedmioty na które uczęszcza, zapisane w formie listy. Następnie zmień dowolny przedmiot na inny. Wyświetl zawartość krotki przed i po zmianach.

## Dodatek

**Zadanie 2.1.1.**

|  |
| --- |
| [module\_2\_lesson\_1\_2\_1\_1.py] |

[module\_2\_lesson\_1\_2\_1\_1.png]

**Zadanie 2.1.2.**

|  |
| --- |
| [module\_2\_lesson\_1\_2\_1\_2.py] |

[module\_2\_lesson\_1\_2\_1\_2.png]

**Zadanie 2.1.3.**

|  |
| --- |
| [module\_2\_lesson\_1\_2\_1\_3.py] |

[module\_2\_lesson\_1\_2\_1\_3.png]

**Zadanie 2.1.4.**

|  |
| --- |
| [module\_2\_lesson\_1\_2\_1\_4.py] |

[module\_2\_lesson\_1\_2\_1\_4.png]

**Zadanie 2.1.5.**

|  |
| --- |
| [module\_2\_lesson\_1\_2\_1\_5.py] |

[module\_2\_lesson\_1\_2\_1\_5.png]

**Zadanie 2.2.1.**

|  |
| --- |
| [module\_2\_lesson\_2\_2\_2\_1.py] |

[module\_2\_lesson\_2\_2\_2\_1.png]

**Zadanie 2.2.2.**

|  |
| --- |
| [module\_2\_lesson\_2\_2\_2\_2.py] |

[module\_2\_lesson\_2\_2\_2\_2.png]

**Zadanie 2.2.3.**

|  |
| --- |
| [module\_2\_lesson\_2\_2\_2\_3.py] |

[module\_2\_lesson\_2\_2\_2\_3.png]

**Zadanie 2.2.4.**

|  |
| --- |
| [module\_2\_lesson\_2\_2\_2\_4.py] |

[module\_2\_lesson\_2\_2\_2\_4.png]

**Zadanie 2.2.5.**

|  |
| --- |
| [module\_2\_lesson\_2\_2\_2\_5.py] |

[module\_2\_lesson\_2\_2\_2\_5.png]

**Zadanie 2.4.1.**

|  |
| --- |
| [module\_2\_summary\_2\_4\_1.py] |

[module\_2\_summary\_2\_4\_1.png]

**Zadanie 2.4.2.**

|  |
| --- |
| [module\_2\_summary\_2\_4\_2.py] |

[module\_2\_summary\_2\_4\_2.png]

**Zadanie 2.4.3.**

|  |
| --- |
| [module\_2\_summary\_2\_4\_3.py] |

[module\_2\_summary\_2\_4\_3.png]

**Zadanie 2.4.4.**

|  |
| --- |
| [module\_2\_summary\_2\_4\_4.py] |

[module\_2\_summary\_2\_4\_4.png]

**Zadanie 2.4.5.**

|  |
| --- |
| [module\_2\_summary\_2\_4\_5.py] |

[module\_2\_summary\_2\_4\_5.png]

**Zadanie 2.4.6.**

|  |
| --- |
| [module\_2\_summary\_2\_4\_6.py] |

[module\_2\_summary\_2\_4\_6.png]

**Zadanie 2.4.7.**

|  |
| --- |
| [module\_2\_summary\_2\_4\_7.py] |

[module\_2\_summary\_2\_4\_7.png]

**Zadanie 2.4.8.**

|  |
| --- |
| [module\_2\_summary\_2\_4\_8.py] |

[module\_2\_summary\_2\_4\_8.png]

**Zadanie 2.4.9.**

|  |
| --- |
| [module\_2\_summary\_2\_4\_9.py] |

[module\_2\_summary\_2\_4\_9.png]

**Zadanie 2.4.10.**

|  |
| --- |
| [module\_2\_summary\_2\_4\_10.py] |

[module\_2\_summary\_2\_4\_10.png]

**Zadanie 2.4.11.**

|  |
| --- |
| [module\_2\_summary\_2\_4\_11.py] |

[module\_2\_summary\_2\_4\_11.png]

**Zadanie 2.4.12.**

|  |
| --- |
| [module\_2\_summary\_2\_4\_12.py] |

[module\_2\_summary\_2\_4\_12.png]

**Zadanie 2.4.13.**

|  |
| --- |
| [module\_2\_summary\_2\_4\_13.py] |

[module\_2\_summary\_2\_4\_13.png]

**Zadanie 2.4.14.**

|  |
| --- |
| [module\_2\_summary\_2\_4\_14.py] |

[module\_2\_summary\_2\_4\_14.png]

**Zadanie 2.4.15.**

|  |
| --- |
| [module\_2\_summary\_2\_4\_15.py] |

[module\_2\_summary\_2\_4\_15.png]

**Zadanie 2.4.16.**

|  |
| --- |
| [module\_2\_summary\_2\_4\_16.py] |

[module\_2\_summary\_2\_4\_16.png]

**Zadanie 2.4.17.**

|  |
| --- |
| [module\_2\_summary\_2\_4\_17.py] |

[module\_2\_summary\_2\_4\_17.png]

**Zadanie 2.4.18.**

|  |
| --- |
| [module\_2\_summary\_2\_4\_18.py] |

[module\_2\_summary\_2\_4\_18.png]

**Zadanie 2.4.19.**

|  |
| --- |
| [module\_2\_summary\_2\_4\_19.py] |

[module\_2\_summary\_2\_4\_19.png]

**Zadanie 2.4.20.**

|  |
| --- |
| [module\_2\_summary\_2\_4\_20.py] |

[module\_2\_summary\_2\_4\_20.png]

# Pętle i iteracje

Pętle to podstawowe narzędzia, które umożliwiają tworzenie bardziej uporządkowanego kodu. Umożliwiają automatyczne powtarzanie określonych operacji. Używając pętli, możesz przetwarzać elementy list, wykonywać obliczenia lub sprawdzać warunki w sposób zautomatyzowany, co znacznie przyspiesza pracę i pozwala na pracę z dużymi zbiorami danych.

## Zrozumienie i zastosowanie pętli

Pętle w języku Python to mechanizm umożliwiający wielokrotne wykonywanie określonego fragmentu kodu. Ułatwiają one powtarzalne zadania, eliminując potrzebę ręcznego powielania instrukcji. Python oferuje dwa główne typy pętli:

* Pętla *for:* Umożliwia iterację po elementach sekwencyjnych, takich jak listy, krotki czy ciągi znaków. Dla każdego elementu wykonywany jest zapisany blok kodu
* Pętla *while*: Wykonuje blok kodu tak długo, jak długo spełniony jest określony warunek logiczny.

Pętle są potężnymi narzędziami i są bardzo często wykorzystywane przy przetwarzaniu dużych zbiorów danych, algorytmach sterowania lub procesach komunikacji.

### Składnia i działanie pętli *for* i *while*

Składnia pętli *for* polega na użyciu słowa kluczowego *for*, po którym następuje nazwa zmiennej (do której przypisywana jest wartość aktualnie pobranego elementu z iterowanego obiektu), a następnie słowo kluczowe *in* i iterowany obiekt. Po dwukropku oraz odpowiednim wcięciu znajduje się blok kodu, który jest wykonywany dla każdego elementu.

|  |
| --- |
| for element in iterable:  # blok kodu wykonywany dla każdego elementu |

Algorytm działania pętli for jest następujący:

1. Iteracja: Pętla pobiera element z iterowanego obiektu (iterable) i jego wartość jest zapisywana do zmiennej (element)
2. Blok kodu: Wykonywane są podane instrukcje wewnątrz pętli dla pobranego elementu. Wszelkie operacje są możliwe do wykonania na elemencie który reprezentuje podana zmienna.
3. Zakończenie: Pętla kończy działanie dla bieżącego elementu i przechodzi do następnego, aż do wyczerpania iterowanego obiektu.

|  |
| --- |
| [module\_3\_lesson\_1\_for\_example.py] |

### Zadania iteracyjne z zastosowaniem funkcji

Pętle poprzez swoje zastosowanie można wykorzystywać na wiele sposobów. Iterowanie obiektów sekwencyjnych i operacje na nich to najbardziej klasyczna metoda, np. poprzez sumowanie elementów listy:

|  |
| --- |
| [module\_3\_lesson\_1\_sum\_list\_elem.py] |

lub literowanie znaków w wyrazie.

|  |
| --- |
| [module\_3\_lesson\_1\_word\_spelling.py] |

Można także wykorzystać pętlę, która będzie oczekiwać od użytkownika dokładnie takiego wejścia, jakiego chcemy i nie pozwoli na kontynuowanie działania programu, jeżeli warunek nie zostanie spełniony.

|  |
| --- |
| [module\_3\_lesson\_1\_loop\_user\_input.py] |

Bardziej zaawansowanym zastosowaniem pętli jest wykorzystanie funkcji wbudowanej *enumerate()*, która przyjmuje jako argument iterowalny obiekt i zwraca pary – pierwszy element to indeks pobranego elementu, a drugi to sam element.

|  |
| --- |
| [module\_3\_lesson\_1\_for\_with\_enumerate.py] |

|  |  |
| --- | --- |
|  | Używając funkcji *enumerate()* należy pamiętać o obsłudze dwóch zmiennych wyjściowych. W przeciwnym razie, interpreter zwróci błąd. Jeżeli chcesz zignorować któryś ze zwracanych elementów, możesz w jego miejscu wstawić znak podłogi (\_). |

Można zapytać, czy skoro możliwe jest rozpakowywanie takich struktur, to czy można je również tworzyć w podobny sposób. Odpowiedź brzmi: tak, ponieważ do tego celu służy wbudowana funkcja *zip()*, która jako argument przyjmuje wiele iterowalnych obiektów, oddzielonych przecinkami.

|  |
| --- |
| [module\_3\_lesson\_1\_for\_with\_zip.py] |

|  |  |
| --- | --- |
|  | Używając funkcji *zip()* należy pamiętać aby iterowalne obiekty były zawsze tej samej długości. W przeciwnym razie, funkcja skróci listy do długości najkrótszej z nich. |

Ostatnim przykładem może być wykorzystanie pętli do tworzenia interaktywnych terminali, gdzie w zależności od wprowadzonego wejścia można kontynuować działanie programu w nieskończonej pętli.

|  |
| --- |
| [module\_3\_lesson\_1\_endless\_while.py] |

### Stosowanie pętli zagnieżdżonych

Pętle zagnieżdżone w Pythonie to konstrukcje, w których jedna pętla znajduje się wewnątrz innej. Pozwalają one na iterację po wielowymiarowych strukturach danych, takich jak listy zagnieżdżone, macierze czy tabele, oraz na generowanie złożonych wzorów. W pętli zewnętrznej iterujemy po jednym zbiorze elementów, a wewnętrzna, wykonywana dla każdego elementu z pętli zewnętrznej, przetwarza kolejny zbiór danych. Dzięki temu można na przykład przejść przez każdy wiersz macierzy, a następnie przez każdy element w danym wierszu.

### Zadania

**Zadanie 3.1.1. Liczby parzyste z zakresu**

Użyj pętli *for* z funkcją *range()*, aby:

* Wypisać wszystkie parzyste liczby od 0 do 20.
* Każdą liczbę wyświetl w osobnej linii.

**Zadanie 3.1.2. Odwracanie kolejności znaków**

Napisz program, który:

* Przy pomocy pętli for odwróci podany łańcuch znaków. Na przykład, dla wejścia "Python" wynik powinien być "nohtyP".

**Zadanie 3.1.3. Wybieranie liczb**

Napisz program, w którym:

* Utworzysz listę zawierającą liczby dodatnie, ujemne i zero.
* Iterujesz po liście i wypisuje tylko liczby dodatnie.

**Zadanie 3.1.4. \* Zgadywanie liczby**

Utwórz program, który:

* Losowo wybiera liczbę z określonego zakresu (np. od 1 do 50) (podpowiedź: użyj funkcji *random.randint()*
* Następnie w pętli while użytkownik próbuje zgadnąć tę liczbę.
* Po każdej próbie program podpowiada, czy zgadywana liczba jest za wysoka, za niska czy poprawna.

**Zadanie 3.1.5. \* Generowanie wzoru w terminalu**

Napisz program, który:

* Za pomocą zagnieżdżonych pętli for (lub for i while) wygeneruje wzór, trójkąta z gwiazdek.
* Użytkownik powinien podać liczbę wierszy, a program wypisze odpowiedni wzór.

## Rekurencja i zasięg zmiennych

Rekurencja i zasięg zmiennych to dwa kluczowe zagadnienia, które warto zrozumieć, aby pisać prawidłowy i efektywny kod w języku Python. Rekurencja niejednokrotnie może stanowić rozwiązanie problemu algorytmicznego przy sortowaniu danych, obliczeniach matematycznych czy kombinatoryce. W tym zadaniu pomoże również zrozumienie zasięgu zmiennych i długości ich życia.

### Zrozumienie rekurencji i tworzenia funkcji rekurencyjnych

Rekurencja to technika programowania, w której funkcja wywołuje samą siebie, aby rozwiązać określony problem. Polega ona na dzieleniu problemu na mniejsze, podobne podproblemy, aż do osiągnięcia najprostszego przypadku – warunku zakończenia. Warunek ten jest kluczowy, ponieważ zapobiega nieskończonemu wywoływaniu funkcji, co mogłoby prowadzić do błędów w programie lub wyczerpania zasobów komputera.

Metoda ta znajduje zastosowanie głównie przy rozwiązywaniu problemów matematycznych oraz w algorytmach o strukturze drzewa. Najbardziej znane przykłady to obliczanie silni liczby oraz wyznaczanie n-tego wyrazu ciągu Fibonacciego.

|  |
| --- |
| [module\_3\_lesson\_2\_fibonacci.py] |

Jednak bardziej realnym problemem programistycznym który może być w ten sposób rozwiązany, jest przetworzenie zagnieżdżonych w sobie struktur danych.

|  |
| --- |
| [module\_3\_lesson\_2\_nested\_list.py] |

### Zmienne globalne a zmienne lokalne

Zmienne w Pythonie są tworzone w momencie przypisania im wartości. Pozostają aktywne tak długo, jak długo program utrzymuje odwołania do przypisanego obiektu. Gdy ostatnie odwołanie do obiektu zostanie usunięte lub zmienna przestanie być dostępna (na przykład, gdy wychodzi poza zakres funkcji), obiekt jest automatycznie usuwany przez mechanizm zwalniania pamięci, zwany *garbage collectorem*, a zmienna przestaje istnieć. Zmienne danego typu i zakres ich życia można opisać następująco:

**Zmienna lokalna:**

* Jest tworzona wewnątrz funkcji lub bloku kodu.
* Jej zakres (ang. scope) ogranicza się do tej funkcji, co oznacza, że poza nią nie jest dostępna.
* Zmienne lokalne są tworzone przy wywołaniu funkcji i znikają po jej zakończeniu, chyba że są zwracane lub przekazywane dalej.

**Zmienna globalna:**

* Definiowana jest poza funkcjami, zazwyczaj na początku modułu.
* Jej zakres obejmuje cały moduł, dzięki czemu jest dostępna w każdej części kodu tego modułu.
* Aby zmodyfikować zmienną globalną wewnątrz funkcji, należy użyć słowa kluczowego *global*, co informuje Pythona, że chcemy pracować z tą globalną zmienną, a nie tworzyć nową zmienną lokalną o tej samej nazwie.

|  |
| --- |
| [module\_3\_lesson\_2\_global\_local\_var.py] |

### Zadania

**Zadanie 3.2.1. Rekurencyjna silnia**

Napisz rekurencyjną funkcję, która:

* Oblicza silnię danej liczby.
* Użyj zmiennych lokalnych do przechowywania wyników pośrednich.

**Zadanie 3.2.2. Rekurencyjne sumowanie**

Napisz funkcję, która:

* Rekurencyjnie sumuje wszystkie elementy podanej listy liczb.
* Każdy poziom rekurencji powinien korzystać z lokalnej zmiennej przechowującej bieżącą sumę.

**Zadanie 3.2.3. Użycie słówka *global***

Utwórz program, który:

* Definiuje globalną zmienną (np. global\_counter) i przypisuje jej początkową wartość.
* W funkcji próbuj modyfikować tę zmienną bez użycia słowa kluczowego global, a następnie z użyciem global, aby pokazać różnicę.
* Wyświetl wartości zmiennej przed i po wywołaniu funkcji.

**Zadanie 3.2.4. \* Rekurencyjne sumowanie cyfr liczby**

Napisz funkcję rekurencyjną, która:

* Oblicza sumę cyfr podanej liczby.

**Zadanie 3.2.5. \* Rekurencyjne obliczenie największego wspólnego dzielnika**

Napisz funkcję rekurencyjną, która:

* Oblicza największy wspólny dzielnik dwóch liczb.

## Iteracja przez elementy w strukturach danych

Iteracja przez elementy w iterowalnych strukturach danych stanowi fundamentalny mechanizm programowania w Pythonie, umożliwiający efektywne przetwarzanie kolekcji obiektów. W języku tym iterowalność to cecha obiektów pozwalająca na sekwencyjny dostęp do ich elementów, bez konieczności ładowania całej struktury do pamięci

### Wykorzystanie pętli do deparsowania danych w strukturach

Iteracja przez różne typy danych w Pythonie odbywa się głównie przy użyciu pętli *for*, która działa na każdym obiekcie iterowalnym. Oto kilka przykładów, jak można iterować przez różne typy danych:

* **Listy i krotki**:W listach i krotkach każdy element jest dostępny po kolei.

|  |
| --- |
| [module\_3\_lesson\_3\_lists\_and\_typles.py] |

* **Łańcuchy znaków (stringi)**: Iteracja przez łańcuch znaków odbywa się znak po znaku.

|  |
| --- |
| [module\_3\_lesson\_3\_strings.py] |

* **Słowniki**: Domyślnie iteracja po słowniku przebiega po kluczach, ale można użyć metod .values() lub .items(), aby uzyskać dostęp do wartości lub par klucz-wartość.

|  |
| --- |
| [module\_3\_lesson\_3\_dicts.py] |

* **Zbiory (set)**: Zbiory są nieuporządkowane, dlatego iteracja po nich nie gwarantuje zachowania konkretnej kolejności.

|  |
| --- |
| [module\_3\_lesson\_3\_sets.py] |

* **Generatory**: Generatory to iterowalne obiekty, które generują wartości na bieżąco, dzięki czemu można iterować po nich bez konieczności przechowywania całej sekwencji w pamięci.

|  |
| --- |
| [module\_3\_lesson\_3\_generators.py] |

### Wykorzystanie list/dictionary comprahension

List comprehensions to elegancki i zwięzły sposób tworzenia nowych list na podstawie istniejących iterowalnych obiektów. Składnia list comprehension to:

|  |
| --- |
| [new\_value for item in iterable if condition] |

Przykład: Jeśli mamy listę liczb i chcemy utworzyć nową listę z kwadratami tylko liczb parzystych, możemy napisać:

|  |
| --- |
| [module\_3\_lesson\_3\_list\_comprahension.py] |

Podobnie, do iteracji po słownikach używa się dictionary comprehensions, które pozwalają tworzyć nowe słowniki. Składnia wygląda następująco:

|  |
| --- |
| {key\_expression: value\_expression for key, value in dictionary.items() if condition} |

Przykład: Mając słownik z liczbami, możemy utworzyć nowy słownik, w którym każdej wartości przyporządkowany jest jej kwadrat:

|  |
| --- |
| [module\_3\_lesson\_3\_dict\_comprahension.py] |

List comprehensions oraz dictionary comprehensions upraszczają iterację, czyniąc kod bardziej czytelnym i efektywnym, zwłaszcza przy przetwarzaniu danych i tworzeniu nowych struktur na podstawie istniejących.

### Zadania

**Zadanie 3.3.1 Wypisywanie elementów listy**

Napisz program, który:

* Przy użyciu pętli for iteruje przez listę elementów (np. liczb lub napisów) i wypisuje każdy element w osobnej linii.

**Zadanie 3.3.2 Iteracja przez słownik**

Napisz program, który:

* Iteruje przez słownik i wypisuje każdą parę klucz–wartość w formacie: „Klucz: [klucz], Wartość: [wartość]”.

**Zadanie 3.3.3 Filtrowanie listy za pomocą list comprehension**

Napisz program, który:

* Na podstawie listy liczb stworzy nową listę zawierającą tylko liczby parzyste.
* Użyj do tego list comprehension.

**Zadanie 3.3.4 \* Generowanie listy krotek z indeksami**

Napisz program, który:

* Wykorzystuje funkcję enumerate oraz list comprehension do stworzenia listy krotek, gdzie każda krotka zawiera indeks oraz wartość elementu z podanej listy.
* Przykład: Dla listy ['a', 'b', 'c'] wynik powinien być [(0, 'a'), (1, 'b'), (2, 'c')].

**Zadanie 3.3.5 \* Pętla while z warunkiem zakończenia**

Napisz program, który:

* Używa pętli while do iteracji przez listę liczb i dodaje kolejne elementy do nowej listy aż do momentu, gdy napotka liczbę spełniającą określony warunek (np. wartość większa niż 50).
* Po zakończeniu pętli wypisz utworzoną listę.

## Podsumowanie

Po przeanalizowaniu poniższego rozdziału warto zapamiętać, że:

* Rekurencja to technika, w której funkcja wywołuje samą siebie, dzieląc problem na mniejsze, podobne podproblemy.
* Kluczowym elementem jest warunek zakończenia, który zapobiega nieskończonym wywołaniom funkcji.
* Znajduje zastosowanie w problemach matematycznych, algorytmach o strukturze drzewa (np. obliczanie silni, ciągu Fibonacciego) oraz przy przetwarzaniu zagnieżdżonych struktur danych.
* Zmienne lokalne są definiowane wewnątrz funkcji lub bloków kodu i są dostępne tylko w ich obrębie. Żyją przez czas wykonywania funkcji.
* Zmienne globalne są definiowane poza funkcjami i dostępne w całym module. Aby je modyfikować wewnątrz funkcji, trzeba użyć słowa kluczowego global.
* Po zakończeniu zakresu lub utracie ostatnich odwołań do zmiennej, obiekt jest usuwany przez garbage collectora.
* Pętle for i while umożliwiają przetwarzanie danych zawartych w listach, krotkach, słownikach, zbiorach, łańcuchach znaków i generatorach.
* List comprehensions i dictionary comprehensions oferują zwięzły sposób tworzenia nowych struktur danych na podstawie istniejących.

## Zadania podsumowujące moduł

**Zadanie 3.5.1. Obliczanie liczby trójkątnej**

* Napisz rekurencyjną funkcję triangular\_number(n), która zwraca n-tą liczbę trójkątną (czyli sumę liczb od 1 do n).

**Zadanie 3.5.2. Ciąg Fibonacciego z memoizacją**

* Zaimplementuj funkcję fibonacci\_memo(n), która rekurencyjnie oblicza n-ty wyraz ciągu Fibonacciego, korzystając z globalnego słownika jako pamięci podręcznej (memoization).

**Zadanie 3.5.3. Rekurencyjne sumowanie parzystych liczb**

* Napisz funkcję recursive\_even\_sum(lst), która rekurencyjnie sumuje tylko parzyste liczby z podanej listy. Użyj lokalnych zmiennych do przechowywania sumy.

**Zadanie 3.5.4. Spłaszczanie zagnieżdżonej listy**

* Utwórz funkcję flatten\_list(nested), która rekurencyjnie spłaszcza zagnieżdżoną listę i zwraca nową listę z wszystkimi elementami na jednym poziomie.

**Zadanie 3.5.5.\*** **Rekurencyjne obliczanie iloczynu cyfr liczby**

* Napisz funkcję recursive\_digit\_product(n), która rekurencyjnie oblicza iloczyn cyfr liczby n. Upewnij się, że funkcja ma prawidłowy warunek zakończenia**.**

**Zadanie 3.5.6. Lokalny i globalny licznik wywołań funkcji**

* Utwórz funkcję call\_counter(), która zwiększa globalny licznik przy każdym wywołaniu oraz zwraca aktualny wynik. Zademonstruj działanie funkcji, wyświetlając licznik przed i po kilku wywołaniach.

**Zadanie 3.5.7. \* Rekurencyjne obliczenie cyfrowego pierwiastka (digital root)**

* Napisz funkcję digital\_root(n), która rekurencyjnie sumuje cyfry liczby aż do uzyskania jednocyfrowego wyniku.

**Zadanie 3.5.8. Filtrowanie palindromów przy użyciu list comprehension**

* Napisz funkcję filter\_palindromes(words), która przyjmuje listę słów i zwraca nową listę zawierającą tylko te słowa, które są palindromami.

**Zadanie 3.5.9. Tworzenie słownika długości słów (dictionary comprehension)**

* Napisz funkcję words\_length\_dict(words), która na podstawie listy słów tworzy słownik, w którym kluczem jest słowo, a wartością jego długość.

**Zadanie 3.5.10.** **Interaktywne sprawdzanie liczby pierwszej (pętla while)**

* Napisz funkcję ask\_prime(), która w pętli while prosi użytkownika o podanie liczby, a następnie sprawdza, czy liczba jest pierwsza. Pętla kończy się, gdy użytkownik poda liczbę pierwszą.

**Zadanie 3.5.11. Generowanie tabliczki mnożenia (zagnieżdżone pętle)**

* Utwórz funkcję multiplication\_table(n), która za pomocą zagnieżdżonych pętli for tworzy tabliczkę mnożenia dla liczb od 1 do n i zwraca wynik jako listę list

**Zadanie 3.5.12.** **Sumowanie elementów listy za pomocą pętli while**

* Napisz funkcję while\_sum(lst), która iteracyjnie (pętlą while) sumuje elementy listy i zwraca wynik.

**Zadanie 3.5.13. Iteracja po krotkach z użyciem enumerate**

* Utwórz funkcję enumerate\_tuples(tuples\_list), która przy użyciu funkcji enumerate iteruje po liście krotek i wypisuje indeks oraz zawartość każdej krotki.

**Zadanie 3.5.14. Zliczanie wystąpień danego słowa w liście zdań**

* Napisz funkcję word\_occurrences(sentences, word), która iteracyjnie zlicza, ile razy dane słowo występuje w liście zdań.

**Zadanie 3.5.15. \* Rekurencyjne przeszukiwanie zagnieżdżonej listy dla konkretnego elementu**

* Napisz funkcję recursive\_find(lst, target), która rekurencyjnie przeszukuje zagnieżdżoną listę i zwraca True, jeśli target zostanie znaleziony, lub False w przeciwnym razie.

**Zadanie 3.5.16. Łączenie listy znaków w łańcuch (pętla for)**

* Utwórz funkcję chars\_to\_string(chars), która iteracyjnie łączy listę pojedynczych znaków w jeden ciąg i zwraca wynik.

**Zadanie 3.5.17. Tworzenie listy kwadratów przy użyciu list comprehension**

* Napisz funkcję squares(n), która przy użyciu list comprehension tworzy listę kwadratów liczb od 1 do n.

**Zadanie 3.5.18. Podwajanie wartości w słowniku (dictionary comprehension)**

* Utwórz funkcję double\_values(d), która przy użyciu dictionary comprehension tworzy nowy słownik, gdzie każda wartość jest podwojona.

**Zadanie 3.5.19. Rekurencyjne obliczenie n-tego wyrazu ciągu z zadanym wzorem**

* Zdefiniuj ciąg, gdzie a(0)=2, a(n)=3\*a(n-1)+1. Napisz rekurencyjną funkcję custom\_sequence(n), która oblicza n-ty wyraz ciągu, korzystając z lokalnych zmiennych.

**Zadanie 3.5.20. \* Demonstracja użycia nonlocal**

* Napisz funkcję outer() zawierającą zmienną lokalną oraz zagnieżdżoną funkcję inner(), która modyfikuje tę zmienną przy użyciu słowa kluczowego nonlocal. Funkcja outer() powinna zwracać zmodyfikowaną wartość.

## Dodatek

**Zadanie 3.1.1.**

|  |
| --- |
| [module\_3\_lesson\_1\_3\_1\_1.py] |

[module\_3\_lesson\_1\_3\_1\_1.png]

**Zadanie 3.1.2.**

|  |
| --- |
| [module\_3\_lesson\_1\_3\_1\_2.py] |

[module\_3\_lesson\_1\_3\_1\_2.png]

**Zadanie 3.1.3.**

|  |
| --- |
| [module\_3\_lesson\_1\_3\_1\_3.py] |

[module\_3\_lesson\_1\_3\_1\_3.png]

**Zadanie 3.1.4.**

|  |
| --- |
| [module\_3\_lesson\_1\_3\_1\_4.py] |

[module\_3\_lesson\_1\_3\_1\_4.png]

**Zadanie 3.1.5.**

|  |
| --- |
| [module\_3\_lesson\_1\_3\_1\_5.py] |

[module\_3\_lesson\_1\_3\_1\_5.png]

**Zadanie 3.2.1.**

|  |
| --- |
| [module\_3\_lesson\_2\_3\_2\_1.py] |

[module\_3\_lesson\_2\_3\_2\_1.png]

**Zadanie 3.2.2.**

|  |
| --- |
| [module\_3\_lesson\_2\_3\_2\_2.py] |

[module\_3\_lesson\_2\_3\_2\_2.png]

**Zadanie 3.2.3.**

|  |
| --- |
| [module\_3\_lesson\_2\_3\_2\_3.py] |

[module\_3\_lesson\_2\_3\_2\_3.png]

**Zadanie 3.2.4.**

|  |
| --- |
| [module\_3\_lesson\_2\_3\_2\_4.py] |

[module\_3\_lesson\_2\_3\_2\_4.png]

**Zadanie 3.2.5.**

|  |
| --- |
| [module\_3\_lesson\_2\_3\_2\_5.py] |

[module\_3\_lesson\_2\_3\_2\_5.png]

**Zadanie 3.3.1.**

|  |
| --- |
| [module\_3\_lesson\_3\_3\_3\_1.py] |

[module\_3\_lesson\_3\_3\_3\_1.png]

**Zadanie 3.3.2.**

|  |
| --- |
| [module\_3\_lesson\_3\_3\_3\_2.py] |

[module\_3\_lesson\_3\_3\_3\_2.png]

**Zadanie 3.3.3.**

|  |
| --- |
| [module\_3\_lesson\_3\_3\_3\_3.py] |

[module\_3\_lesson\_3\_3\_3\_3.png]

**Zadanie 3.3.4.**

|  |
| --- |
| [module\_3\_lesson\_3\_3\_3\_4.py] |

[module\_3\_lesson\_3\_3\_3\_4.png]

**Zadanie 3.3.5.**

|  |
| --- |
| [module\_3\_lesson\_3\_3\_3\_5.py] |

[module\_3\_lesson\_3\_3\_3\_5.png]

**Zadanie 3.5.1.**

|  |
| --- |
| [module\_3\_summary\_3\_5\_1.py] |

[module\_3\_summary\_3\_5\_1.png]

**Zadanie 3.5.2.**

|  |
| --- |
| [module\_3\_summary\_3\_5\_2.py] |

[module\_3\_summary\_3\_5\_2.png]

**Zadanie 3.5.3.**

|  |
| --- |
| [module\_3\_summary\_3\_5\_3.py] |

[module\_3\_summary\_3\_5\_3.png]

**Zadanie 3.5.4.**

|  |
| --- |
| [module\_3\_summary\_3\_5\_4.py] |

[module\_3\_summary\_3\_5\_4.png]

**Zadanie 3.5.5.**

|  |
| --- |
| [module\_3\_summary\_3\_5\_5.py] |

[module\_3\_summary\_3\_5\_5.png]

**Zadanie 3.5.6.**

|  |
| --- |
| [module\_3\_summary\_3\_5\_6.py] |

[module\_3\_summary\_3\_5\_6.png]

**Zadanie 3.5.7.**

|  |
| --- |
| [module\_3\_summary\_3\_5\_7.py] |

[module\_3\_summary\_3\_5\_7.png]

**Zadanie 3.5.8.**

|  |
| --- |
| [module\_3\_summary\_3\_5\_8.py] |

[module\_3\_summary\_3\_5\_8.png]

**Zadanie 3.5.9.**

|  |
| --- |
| [module\_3\_summary\_3\_5\_9.py] |

[module\_3\_summary\_3\_5\_9.png]

**Zadanie 3.5.10.**

|  |
| --- |
| [module\_3\_summary\_3\_5\_10.py] |

[module\_3\_summary\_3\_5\_10.png]

**Zadanie 3.5.11.**

|  |
| --- |
| [module\_3\_summary\_3\_5\_11.py] |

[module\_3\_summary\_3\_5\_11.png]

**Zadanie 3.5.12.**

|  |
| --- |
| [module\_3\_summary\_3\_5\_12.py] |

[module\_3\_summary\_3\_5\_12.png]

**Zadanie 3.5.13.**

|  |
| --- |
| [module\_3\_summary\_3\_5\_13.py] |

[module\_3\_summary\_3\_5\_13.png]

**Zadanie 3.5.14.**

|  |
| --- |
| [module\_3\_summary\_3\_5\_14.py] |

[module\_3\_summary\_3\_5\_14.png]

**Zadanie 3.5.15.**

|  |
| --- |
| [module\_3\_summary\_3\_5\_15.py] |

[module\_3\_summary\_3\_5\_15.png]

**Zadanie 3.5.16.**

|  |
| --- |
| [module\_3\_summary\_3\_5\_16.py] |

[module\_3\_summary\_3\_5\_16.png]

**Zadanie 3.5.17.**

|  |
| --- |
| [module\_3\_summary\_3\_5\_17.py] |

[module\_3\_summary\_3\_5\_17.png]

**Zadanie 3.5.18.**

|  |
| --- |
| [module\_3\_summary\_3\_5\_18.py] |

[module\_3\_summary\_3\_5\_18.png]

**Zadanie 3.5.19.**

|  |
| --- |
| [module\_3\_summary\_3\_5\_19.py] |

[module\_3\_summary\_3\_5\_19.png]

**Zadanie 3.5.20.**

|  |
| --- |
| [module\_3\_summary\_3\_5\_20.py] |

[module\_3\_summary\_3\_5\_20.png]

# Funkcje i moduły

Jednym z elementów dobrego stylu programowania jest odpowiednia dokumentacja kodu i jego dobra organizacja. Jeśli chodzi o dokumentowanie, w języku Python stosuje się do tego docstringi, czyli specjalne komentarze opisujące działanie funkcji, ich parametry oraz zwracane wartości. W tym rozdziale przyjrzymy się podstawowej składni definiowania funkcji w Pythonie oraz przykładom ich dokumentacji. Nauczymy się, jak tworzyć funkcje przyjmujące argumenty oraz zwracające wartości, co pozwala na lepszą modularność kodu.

## Zastosowanie funkcji

Funkcja to jedno z najważniejszych narzędzi, które występują w języku Python. Może to ilustrować fakt, że istnieje cały paradygmat programowania, nazywany programowaniem funkcyjnym. Funkcja składa się z nazwy oraz ciała, które definiowane jest przez odpowiednie wcięcie. Przyjmuje ona argumenty, czyli zmienne które w swoim ciele przetwarza i może te przetworzone wartości zwracać. Podczas definiowania funkcji należy pamiętać, że jej definicja musi znajdować się przed miejscem wywołania.

Funkcje są fundamentalnym elementem każdego popularnego języka programowania. Funkcje zapewniają organizację kodu, jego wielokrotne wykorzystanie oraz zwiększenie czytelności. W tym rozdziale omówimy wykorzystywanie oraz tworzenie funkcji wraz z przekazywaniem argumentów.

W języku Python, funkcję definiuje się przy użyciu słowa kluczowego *def*. Następnie podaje się nazwę funkcji oraz w nawiasach listę parametrów, po których umieszcza się dwukropek. Ciało funkcji, czyli blok instrukcji, musi mieć odpowiednie wcięcie.

Funkcja wypisująca imię

|  |
| --- |
| [module\_4\_lesson\_1\_fun.py] |

Funkcja może zwracać wartość za pomocą słowa kluczowego *return*. Wówczas po tym słowie kluczowym wskazuje się, co ma ona zwrócić. Mogą to być zarówno zmienne dowolnego typu, jak również wynik operacji.

Funkcja zwracająca wartości

|  |
| --- |
| [module\_4\_lesson\_1\_fun\_return.py] |

**Dokumentacja**  
Jednym z elementów dobrego stylu programowania jest odpowiednia dokumentacja kodu. W Pythonie stosuje się do tego docstringi, czyli specjalne komentarze opisujące działanie funkcji, ich parametry oraz zwracane wartości.

|  |
| --- |
| [module\_4\_lesson\_1\_docs.py] |

**Lambdy**

Funkcje anonimowe *lambda()*, pozwalają na tworzenie krótkich, jednowierszowych funkcji. Są one przydatne w sytuacjach, gdy nie ma potrzeby definiowania pełnej funkcji z nazwą.

Obliczenie kwadratu oraz sumy liczb za pomocą lambdy

|  |
| --- |
| [module\_4\_lesson\_1\_lambda.py] |

**Argumenty**   
Mechanizmy \*args i \*\*kwargs pozwalają na przekazywanie dowolnej liczby argumentów do funkcji. Poznamy ich zastosowanie w praktycznych przykładach.

Przekazywanie args oraz kwargs do funkcji

|  |
| --- |
| [module\_4\_lesson\_1\_arguments.py] |

### Zadania

**Zadanie 4.1.1. Prosta funkcja**

* Napisz funkcję pole\_prostokata, która przyjmuje dwa argumenty: a i b, a następnie zwraca pole prostokąta.

**Zadanie 4.1.2. Dokumentacja funkcji**

* Napisz funkcję oblicz\_potęge, która podnosi liczbę x do potęgi y i posiada odpowiednią dokumentację docstring.

**Zadanie 4.1.3. Funkcja lambda**

* Zdefiniuj funkcję maksimum w postaci wyrażenia lambda, która zwróci większą z dwóch liczb.

**Zadanie 4.1.4. Wykorzystanie args**

* Napisz funkcję suma\_elementow, która przyjmuje dowolną liczbę argumentów i zwraca ich sumę.

**Zadanie 4.1.5. Wykorzystanie kwargs**

* Napisz funkcję dane\_osobowe, która przyjmuje dowolne argumenty nazwane i wyświetla je w formacie klucz: wartość.

## Popularne moduły (biblioteki)

Język Python posiada bogaty zbiór bibliotek standardowych i zewnętrznych. W tej sekcji omówimy najczęściej wykorzystywane moduły oraz sposoby ich importowania i używania.

Popularne moduły wbudowane wraz z opisem

|  |  |
| --- | --- |
| **Moduł** | **Opis** |
| os | Operacje systemowe. |
| sys | Interakcja ze środowiskiem wykonawczym. |
| math | Funkcje matematyczne. |
| random | Generowanie liczb losowych. |
| datetime | Operacje na datach i czasie. |
| json | Obsługa danych w formacie JSON. |

|  |  |
| --- | --- |
|  | Aby sprawdzić, jakie funkcje i klasy znajdują się w danym module, można użyć funkcji dir(). |

**Biblioteka os**

Moduł umożliwia wykonywanie operacji systemowych, takich jak zarządzanie plikami i katalogami. Pozwala na interakcję z systemem operacyjnym, np. pobieranie zmiennych środowiskowych. Umożliwia także uruchamianie procesów i manipulowanie ścieżkami plików.

**Biblioteka sys**

Zapewnia dostęp do zmiennych i funkcji związanych ze środowiskiem wykonawczym Pythona. Pozwala na manipulowanie strumieniami wejścia i wyjścia oraz zakończenie programu. Umożliwia także dostęp do argumentów przekazanych do skryptu.

**Biblioteka math**

Moduł dostarcza funkcje matematyczne, takie jak pierwiastkowanie, potęgowanie i trygonometria. Umożliwia wykonywanie precyzyjnych obliczeń numerycznych. Zawiera również stałe matematyczne, np. π (pi) i e (liczba Eulera).

**Biblioteka random**

Służy do generowania liczb losowych i losowego wyboru elementów z sekwencji. Umożliwia m.in. symulację rzutów kostką, tasowanie list i wybieranie losowych wartości. Obsługuje także różne rozkłady probabilistyczne.

**Biblioteka datetime**

Pozwala na operacje związane z datami i czasem, takie jak obliczenia różnicy czasu czy formatowanie dat. Umożliwia pobieranie aktualnej daty i godziny. Obsługuje również konwersję między różnymi formatami daty i czasu.

**Biblioteka json**

Moduł umożliwia konwersję danych między formatem JSON a strukturami Pythona. Przydaje się do zapisu i odczytu danych w aplikacjach internetowych oraz API. Obsługuje zarówno serializację, jak i deserializację danych JSON.

### Zadania

**Zadanie 4.2.1 Użycie funkcji matematycznych**

* Napisz program wykorzystujący moduł math, który obliczy pierwiastek kwadratowy z liczby podanej przez użytkownika.

**Zadanie 4.2.2. Generowanie liczb losowych**

* Napisz skrypt wykorzystujący moduł random, który wylosuje 5 liczb z zakresu 1-100 i wyświetli je w posortowanej kolejności.

**Zadanie 4.2.3. Importowanie i użycie różnych bibliotek**

* Napisz skrypt, który wykorzysta trzy różne biblioteki (math, random, datetime) do wykonania podstawowych operacji.

**Zadanie 4.2.4. Instalacja i sprawdzenie dostępności biblioteki**

* Napisz skrypt, który sprawdzi, czy biblioteka requests jest zainstalowana, a jeśli nie, to wyświetli komunikat o konieczności jej instalacji.

**Zadanie 4.2.5. \* Lista dostępnych bibliotek w środowisku**

* Napisz program, który wyświetli listę wszystkich zainstalowanych bibliotek w aktualnym środowisku za pomocą biblioteki pkg\_resources.

## Tworzenie własnych modułów

Język Python umożliwia organizowanie kodu w moduły, co ułatwia jego ponowne użycie i zarządzanie większymi projektami. W Pythonie moduły umożliwiają podział kodu na logiczne sekcje, które mogą być importowane i wykorzystywane w innych skryptach.

Moduły to po prostu pliki .py, które zawierają definicje funkcji, klas i zmiennych. W tej sekcji omówimy, jak tworzyć i organizować moduły oraz jak korzystać z nich w innych skryptach.

Tworzenie pliku modułu moj\_modul.py

|  |
| --- |
| [module\_4\_lesson\_2\_moj\_modul.py] |

Importowanie modułów

|  |
| --- |
| [module\_4\_lesson\_2\_moj\_modul\_import.py] |

Moduły organizowane są w pakiety. Inaczej mówiąc, pakiet składa się z wielu modułów z dodatkowym plikiem *\_\_init\_\_.py* w folderze. Plik ten może być pozostawiony pusty. Podczas pracy z modułami i pakietami w większych projektach warto zarządzać zależnościami. Można to zrobić, zapisując wymagane moduły w pliku requirements.txt.

### Zadania

**Zadanie 4.3.1. Tworzenie prostego modułu**

* Napisz moduł kalkulator.py, który zawiera funkcję suma(a, b), zwracającą sumę dwóch liczb.

**Zadanie 4.3.2. Importowanie funkcji z modułu**

* Stwórz skrypt, który zaimportuje funkcję suma z modułu z poprzedniego zadania i użyje jej do dodania dwóch liczb podanych przez użytkownika.

**Zadanie 4.3.3. Tworzenie pakietu**

* Utwórz pakiet *tools* zawierający moduły kalkulator.py i konwerter.py. Każdy moduł powinien zawierać jedną funkcję matematyczną. Zaimportuje je i wykorzystaj,

**Zadanie 4.3.4. Użycie init.py w pakiecie**

* Dodaj plik init.py do pakietu tools, który zaimportuje funkcje ze wszystkich modułów pakietu tak jak w poprzednim zadaniu. Wykorzystaj funkcje modułów jeszcze raz.

Podpowiedź importu: from tools import dodaj, celsius\_na\_fahrenheit

**Zadanie 4.3.5. \* Tworzenie i zapisywanie zależności**

* Napisz skrypt, który generuje plik requirements.txt z listą na podstawie zainstalowanych pakietów w środowisku Python. Wykorzystaj bibliotekę subprocess.

## Podsumowanie

Ten moduł omówił tworzenie funkcji i modułów w Pythonie. Nauczyliśmy się, jak tworzyć i dokumentować funkcje, wykorzystując docstringi, funkcje anonimowe lambda(), a także deklarować argumenty i typy zwracanych wartości. Poznaliśmy również sposób użycia \*args i \*\*kwargs w funkcjach.

Dodatkowo zapoznaliśmy się z najczęściej wykorzystywanymi modułami wbudowanymi, takimi jak os, sys, math, random, datetime i json, a także z popularnymi bibliotekami zewnętrznymi, w tym numpy, pandas, matplotlib, pygame i opencv.

Na koniec nauczyliśmy się tworzyć własne moduły, organizować ich strukturę za pomocą init.py, importować je oraz zapisywać zależności w pliku requirements.txt.

## Zadania podsumowujące moduł

**Zadanie 4.5.1. Liczenie znaków**

* Napisz funkcję count\_chars(text), która zwraca liczbę znaków w podanym ciągu.

**Zadanie 4.5.2. Obliczanie logarytmu**

* Napisz funkcję oblicz\_logarytm(x, podstawa), która oblicza logarytm liczby x przy zadanej podstawie wykorzystując funkcję math.log(), przekazując do niej tylko jeden parametr. Funkcja powinna zawierać odpowiednią dokumentację (docstring).

Podpowiedź:

**Zadanie 4.5.3. Odwracanie tekstu**

* Napisz funkcję reverse\_text(text), która zwraca odwrócony tekst.

**Zadanie 4.5.4. Silnia iteracyjnie**

* Napisz funkcję factorial(n), która oblicza silnię liczby n za pomocą pętli for.

**Zadanie 4.5.5. Odwrócenie kolejności elementów listy**

* Napisz funkcję, która jako parametr przyjmie listę, a następnie zwróci ją z elementami w odwróconej kolejności

**Zadanie 4.5.6. Liczba parzysta czy nieparzysta**

* Napisz funkcję is\_even(n), która sprawdza, czy liczba n jest parzysta.

**Zadanie 4.5.7. Kwadrat liczby**

* Napisz funkcję lambda square(x), która zwraca kwadrat liczby x.

**Zadanie 4.5.8. Minimum z dwóch liczb**

* Napisz funkcję lambda min\_value(a,b), która zwraca minimum z podanych liczb

**Zadanie 4.5.9. \* Sprawdzanie czy liczba jest pierwsza**

* Napisz funkcję, która jako parametr przyjmie liczbę a następnie sprawdzi czy jest liczbą pierwszą zwracając *True* lub *False.*

**Zadanie 4.5.10. \* Manipulowanie na wartościach zawartych w listach**

* Napisz funkcję, która jako argumenty przyjmie dwie, dwuwymiarowe listy o tym samym rozmiarze. Następnie doda do siebie odpowiadające sobie elementy i ich sumę podzieli przez 3. Wyświetl wyniki tych operacji jako osobną listę.

**Zadanie 4.5.11. Pobieranie pierwszego znaku**

* Napisz funkcję lambda first\_char(s), która zwraca pierwszy znak napisu.

**Zadanie 4.5.12. Łączenie dwóch napisów**

* Napisz funkcję lambda concat(s1, s2), która zwraca połączone dwa napisy z separatorem spacji.

**Zadanie 4.5.13. Obliczanie pierwiastka kwadratowego**

* Napisz funkcję sqrt\_value(n), która zwraca pierwiastek kwadratowy liczby n przy użyciu math.sqrt().

**Zadanie 4.5.14. Losowanie liczby**

* Napisz funkcję random\_number(), która zwraca losową liczbę z zakresu 1-100 przy użyciu random.randint().

**Zadanie 4.5.15. Analiza częstości znaków w tekście**

* Napisz funkcję char\_frequency(text), która zwraca słownik z częstością występowania znaków w tekście (ignoruje wielkość liter).

**Zadanie 4.5.16. Obliczanie średniej z listy**

* Napisz funkcję mean\_value(lst), która zwraca średnią arytmetyczną listy liczb przy użyciu numpy.mean().

**Zadanie 4.5.17. Sortowanie listy liczb**

* Napisz funkcję sort\_list(lst), która zwraca posortowaną listę liczb przy użyciu sorted().

**Zadanie 4.5.18. Pomiar czasu wykonania kodu**

* Napisz funkcję measure\_time(), która mierzy czas wykonania operacji przy użyciu time.time().

**Zadanie 4.5.19. Wyszukiwanie unikalnych elementów w liście**

* Napisz funkcję unique\_elements(lst), która zwraca listę unikalnych elementów.

**Zadanie 4.5.20. Znajdowanie wartości największej i najmniejszej w liście**

* Napisz funkcję min\_max(lst), która zwraca krotkę zawierającą najmniejszy i największy element listy.

## Dodatek

**Zadanie 4.1.1.**

|  |
| --- |
| [module\_4\_lesson\_1\_4\_1\_1.py] |

[module\_4\_lesson\_1\_4\_1\_1.png]

**Zadanie 4.1.2.**

|  |
| --- |
| [module\_4\_lesson\_1\_4\_1\_2.py] |

[module\_4\_lesson\_1\_4\_1\_2.png]

**Zadanie 4.1.3.**

|  |
| --- |
| [module\_4\_lesson\_1\_4\_1\_3.py] |

[module\_4\_lesson\_1\_4\_1\_3.png]

**Zadanie 4.1.4.**

|  |
| --- |
| [module\_4\_lesson\_1\_4\_1\_4.py] |

[module\_4\_lesson\_1\_4\_1\_4.png]

**Zadanie 4.1.5.**

|  |
| --- |
| [module\_4\_lesson\_1\_4\_1\_5.py] |

[module\_4\_lesson\_1\_4\_1\_5.png]

**Zadanie 4.2.1.**

|  |
| --- |
| [module\_4\_lesson\_2\_4\_2\_1.py] |

[module\_4\_lesson\_2\_4\_2\_1.png]

**Zadanie 4.2.2.**

|  |
| --- |
| [module\_4\_lesson\_2\_4\_2\_2.py] |

[module\_4\_lesson\_2\_4\_2\_2.png]

**Zadanie 4.2.3.**

|  |
| --- |
| [module\_4\_lesson\_2\_4\_2\_3.py] |

[module\_4\_lesson\_2\_4\_2\_3.png]

**Zadanie 4.2.4.**

|  |
| --- |
| [module\_4\_lesson\_2\_4\_2\_4.py] |

[module\_4\_lesson\_2\_4\_2\_4.png]

**Zadanie 4.2.5.**

|  |
| --- |
| [module\_4\_lesson\_2\_4\_2\_5.py] |

[module\_4\_lesson\_2\_4\_2\_5.png]

**Zadanie 4.5.1.**

|  |
| --- |
| [module\_4\_summary\_4\_5\_1.py] |

[module\_4\_summary\_4\_5\_1.png]

**Zadanie 4.5.2.**

|  |
| --- |
| [module\_4\_summary\_4\_5\_2.py] |

[module\_4\_summary\_4\_5\_2.png]

**Zadanie 4.5.3.**

|  |
| --- |
| [module\_4\_summary\_4\_5\_3.py] |

[module\_4\_summary\_4\_5\_3.png]

**Zadanie 4.5.4.**

|  |
| --- |
| [module\_4\_summary\_4\_5\_4.py] |

[module\_4\_summary\_4\_5\_4.png]

**Zadanie 4.5.5.**

|  |
| --- |
| [module\_4\_summary\_4\_5\_5.py] |

[module\_4\_summary\_4\_5\_5.png]

**Zadanie 4.5.6.**

|  |
| --- |
| [module\_4\_summary\_4\_5\_6.py] |

[module\_4\_summary\_4\_5\_6.png]

**Zadanie 4.5.7.**

|  |
| --- |
| [module\_4\_summary\_4\_5\_7.py] |

[module\_4\_summary\_4\_5\_7.png]

**Zadanie 4.5.8.**

|  |
| --- |
| [module\_4\_summary\_4\_5\_8.py] |

[module\_4\_summary\_4\_5\_8.png]

**Zadanie 4.5.9.**

|  |
| --- |
| [module\_4\_summary\_4\_5\_9.py] |

[module\_4\_summary\_4\_5\_9.png]

**Zadanie 4.5.10.**

|  |
| --- |
| [module\_4\_summary\_4\_5\_10.py] |

[module\_4\_summary\_4\_5\_10.png]

**Zadanie 4.5.11.**

|  |
| --- |
| [module\_4\_summary\_4\_5\_11.py] |

[module\_4\_summary\_4\_5\_11.png]

**Zadanie 4.5.12.**

|  |
| --- |
| [module\_4\_summary\_4\_5\_12.py] |

[module\_4\_summary\_4\_5\_12.png]

**Zadanie 4.5.13.**

|  |
| --- |
| [module\_4\_summary\_4\_5\_13.py] |

[module\_4\_summary\_4\_5\_13.png]

**Zadanie 4.5.14.**

|  |
| --- |
| [module\_4\_summary\_4\_5\_14.py] |

[module\_4\_summary\_4\_5\_14.png]

**Zadanie 4.5.15.**

|  |
| --- |
| [module\_4\_summary\_4\_5\_15.py] |

[module\_4\_summary\_4\_5\_15.png]

**Zadanie 4.5.16.**

|  |
| --- |
| [module\_4\_summary\_4\_5\_16.py] |

[module\_4\_summary\_4\_5\_16.png]

**Zadanie 4.5.17.**

|  |
| --- |
| [module\_4\_summary\_4\_5\_17.py] |

[module\_4\_summary\_4\_5\_17.png]

**Zadanie 4.5.18.**

|  |
| --- |
| [module\_4\_summary\_4\_5\_18.py] |

[module\_4\_summary\_4\_5\_18.png]

**Zadanie 4.5.19.**

|  |
| --- |
| [module\_4\_summary\_4\_5\_19.py] |

[module\_4\_summary\_4\_5\_19.png]

**Zadanie 4.5.20.**

|  |
| --- |
| [module\_4\_summary\_4\_5\_20.py] |

[module\_4\_summary\_4\_5\_20.png]

# Klasy, obsługa plików, operacje wejścia/wyjścia, integracja z zewnętrznymi urządzeniami

W tej części kursu poznamy podstawowe pojęcia programowania obiektowego: klasy, obiekty, metody i atrybuty. Nauczymy się tworzyć i używać własne klasy oraz radzić sobie z błędami podczas wczytywania plików. Przećwiczymy operacje na plikach tekstowych i różnych formatach, takich jak *CSV* i *JSON*. Sprawdzimy, jak wykrywać podłączone urządzenia i pracować z danymi na zewnętrznych nośnikach. Na koniec zajmiemy się obsługą urządzeń zewnętrznych, m.in myszki i kamery, przechwytując oraz przetwarzając obraz za pomocą *OpenCV*.

## Wprowadzenie do programowania obiektowego i klas

Programowanie obiektowe (OOP) to paradygmat programowania, który pozwala na organizację kodu “jako obiekty”. Każdy obiekt jest instancją klasy, która określa jego strukturę i zachowanie. Klasy składają się z atrybutów (zmiennych przechowujących stan obiektu) oraz metod (funkcji definiujących operacje na obiekcie). Tworzenie klas w języku Python pozwala na lepszą modularność i czytelność kodu.

Składowe klasy

|  |  |
| --- | --- |
| **Składowa** | **Opis** |
| Atrybuty | Zmienne przechowujące stan obiektu |
| Metody | Funkcje definiujące operacje, które można wykonać na obiekcie. |
| Konstruktor | Specjalna metoda inicjalizująca obiekt podczas jego tworzenia: *\_\_init\_\_()*. |
| Atrybuty klasowe | Zmienne współdzielone przez wszystkie instancje klasy. |
| Metody statyczne i klasy | Funkcje, które mogą działać na poziomie klasy, a nie konkretnego obiektu. |

Klasa w programowaniu to szablon definiujący strukturę i zachowanie obiektów, zawierający atrybuty oraz metody operujące na nich. Służy do tworzenia obiektów, które korzystają z atrybutów i funkcje zdefiniowane w klasie.

Przykładem klasy być klasa Kot. Taka klasa może definiować cechy (np. kolor, wiek, rasa) oraz zachowania (np. miauczenie, jedzenie, spanie), a każdy konkretny kot (np. Mruczek, Puszek) będzie jej instancją.

Klasa z kotami

|  |
| --- |
| [module\_5\_lesson\_1\_cats.py] |

### Zadania

**Zadanie 5.1.1. Klasa Osoba**

* Napisz klasę Osoba, która przechowuje imię i wiek. Dodaj metodę przedstaw\_sie(), która zwróci tekst w formacie: "Cześć, mam na imię IMIĘ i mam WIEK lat."

**Zadanie 5.1.2. Klasa Pies**

* Napisz klasę Pies, która przechowuje imię i rasę psa. Dodaj metodę szczekaj(), która zwróci: "IMIĘ mówi: Hau! Hau!"

**Zadanie 5.1.3. Klasa Prostokąt**

* Napisz klasę Prostokąt, która przyjmuje szerokość i wysokość. Dodaj metodę pole(), która zwróci pole prostokąta.

**Zadanie 5.1.4. Klasa Produkt**

* Napisz klasę Produkt, która przyjmuje nazwę produktu i cenę. Dodaj metodę opis(), która zwróci informację: "Produkt: NAZWA, Cena: CENA zł"

**Zadanie 5.1.5. \* Klasa BankKonto z obsługą salda**

* Napisz klasę BankKonto, która przechowuje numer konta i saldo. Dodaj metody:
  + wplac(kwota) – dodaje kwotę do salda,
  + wyplac(kwota) – odejmuje kwotę, jeśli saldo na to pozwala,
  + wsprawdz\_saldo() – zwraca aktualne saldo w formacie "Saldo: X zł".

## Obsługa popularnych błędów i własnych klas wyjątków

Podczas pracy z danymi mogą wystąpić błędy, np. brak pliku, błędne wartości wejściowe czy problemy z odczytem danych. Język Python umożliwia obsługę błędów za pomocą bloków try-except, które pozwalają na przechwycenie ewentualnych błędów i odpowiednią reakcję programu. Możliwe jest również definiowanie własnych klas wyjątków, co zwiększa kontrolę nad procesem obsługi błędów własnego kodu.

### Popularne błędy

Podczas pracy z danymi w języku Python często można napotkać różne błędy, które mogą prowadzić do niepoprawnego działania programu. Obsługa wyjątków pozwala na wykrywanie takich sytuacji i odpowiednią reakcję na takie zdarzenie. Klasa wyjątku to specjalny typ klasy w języku Python, dziedziczący po klasie Exception, który umożliwia definiowanie własnych, niestandardowych błędów i precyzyjne zarządzanie wyjątkami w programie.

Niektóre wbudowane wyjątki

|  |  |
| --- | --- |
| **Klasa** | **Opis** |
| ValueError | Pojawia się, gdy funkcja otrzyma niewłaściwy typ danych, np. zamiana tekstu na liczbę |
| IndexError | Występuje, gdy próbujemy uzyskać dostęp do elementu poza zakresem listy |
| KeyError | Pojawia się, gdy próbujemy odwołać się do nieistniejącego klucza w słowniku. |
| TypeError | Występuje, gdy operacja jest wykonywana na niewłaściwym typie danych. |
| ZeroDivisionError | Występuje, gdy program próbuje podzielić liczbę przez zero. |

Podczas pracy z plikami warto odpowiednio przygotować program na pewne często spotykane problemy. Najczęściej występujące błędy podczas pracy z plikami dotyczą braku pliku, braku uprawnień, próby otwarcia katalogu zamiast pliku oraz ogólnych problemów wejścia-wyjścia. Sytuacje, w których warto zastosować obsługę wyjątków, to na przykład:

Jeśli aplikacja wymaga specyficznych reguł dla wprowadzanych danych (np. numer musi mieć dokładnie 10 cyfr), można stworzyć własny wyjątek NiepoprawnyNumerError, który zostanie zgłoszony, gdy użytkownik poda błędny numer.

W systemie bankowym można utworzyć wyjątek BrakŚrodkówError, który będzie zgłaszany, gdy użytkownik spróbuje wypłacić więcej pieniędzy, niż ma na koncie.

Wyjątki związane z plikami i obsługą wejścia-wyjścia

|  |  |
| --- | --- |
| **Klasa** | **Opis** |
| FileNotFoundError | Występuje, gdy program próbuje otworzyć plik, który nie istnieje |
| PermissionError | Brak uprawnień do odczytu lub zapisu pliku. |
| IsADirectoryError | Próba otwarcia katalogu zamiast pliku |
| IOError | Ogólny błąd wejścia-wyjścia |

Przykład wyjątków

|  |
| --- |
| [module\_5\_lesson\_2\_errors.py] |

### Własne klasy wyjątków

W Pythonie istnieje możliwość definiowania własnych klas wyjątków, co pozwala na lepszą kontrolę nad specyficznymi błędami w programie. Tworząc własne wyjątki, można precyzyjnie obsługiwać różne ewentualne błędy, które nie są uwzględnione w standardowych wyjątkach języka.

Własny wyjątek

|  |
| --- |
| [module\_5\_lesson\_2\_custom\_exception.py] |

|  |  |
| --- | --- |
|  | Zwróć uwagę, że klasa wyjątku musi dziedziczyć z klasy Exception, co oznacza, że stanie się częścią systemu obsługi wyjątków w Pythonie. |

### Zadania

**Zadanie 5.2.1. Obsługa błędu dzielenia przez zero**

* Napisz program, który pobiera dwie liczby od użytkownika i wykonuje dzielenie pierwszej przez drugą. Obsłuż wyjątek ZeroDivisionError, aby zapobiec błędowi w przypadku próby dzielenia przez zero.

**Zadanie 5.2.2. Obsługa błędu nieistniejącego pliku**

* Napisz program, który próbuje otworzyć plik dane.txt i wypisać jego zawartość. Obsłuż wyjątek FileNotFoundError, aby program nie zakończył działania, jeśli plik nie istnieje.

**Zadanie 5.2.3. Obsługa błędnego indeksu w liście**

* Napisz program, który pobiera od użytkownika indeks i próbuje uzyskać dostęp do elementu listy liczby = [10, 20, 30, 40]. Obsłuż wyjątek IndexError, jeśli użytkownik poda indeks spoza zakresu listy.

**Zadanie 5.2.4. Obsługa błędnego klucza w słowniku**

* Napisz program, który pobiera od użytkownika nazwę kraju i zwraca jego stolicę ze słownika kraje = {"Polska": "Warszawa", "Niemcy": "Berlin", "Francja": "Paryż"}. Obsłuż wyjątek KeyError, jeśli użytkownik poda nazwę kraju, którego nie ma w słowniku.

**Zadanie 5.2.5. \* Własna klasa wyjątku dla pustego pliku**

* Napisz klasę wyjątku PustyPlikError, która zostanie zgłoszona, jeśli program spróbuje otworzyć plik i jego zawartość okaże się pusta. Następnie napisz program, który otwiera plik dane.txt, sprawdza czy zawiera dane, i w przypadku pustego pliku zgłasza wyjątek PustyPlikError.

## Obsługa plików – odczyt i zapis danych

Operacje na plikach tekstowych są nieodłącznym elementem zarządzania danymi w każdym języku programowania. Do odczytu zawartości plików służy funkcja *open()* używana w trybie czytania (oznaczanym literą 'r'), która umożliwia pobranie zawartości pliku linia po linii. Zapisywanie danych do plików można realizować w trybie zapisu (znak w) lub dopisywania (znak a). Podczas pracy z plikami warto stosować obsługę wyjątków, aby unikać błędów związanych np. z nieistniejącym plikiem lub brakiem uprawnień do zapisu.

Tryby obsługi plików

|  |  |
| --- | --- |
| Tryb | Opis |
| r | Otwiera plik do odczytu. Plik musi istnieć. |
| w | Tworzy nowy plik lub nadpisuje istniejący. |
| a | Otwiera plik do dopisywania na końcu. Tworzy plik, jeśli nie istnieje. |
| r+ | Otwiera plik do odczytu i zapisu. Plik musi istnieć. |
| w+ | Tworzy nowy plik do odczytu i zapisu lub nadpisuje istniejący. |
| a+ | Otwiera plik do odczytu i dopisywania. Tworzy plik, jeśli nie istnieje. |

Obsługa plików

|  |
| --- |
| [module\_5\_lesson\_2\_text\_files.py] |

### Zadania

**Zadanie 5.3.1. Odczyt i wyświetlenie zawartości pliku**

* Napisz program, który otworzy plik *log.txt* w trybie r, a następnie wyświetli jego zawartość linia po linii.

**Zadanie 5.3.2. Zapis i nadpisanie zawartości pliku**

* Napisz program, który otworzy plik notatka.txt w trybie w, zapisze w nim trzy linie tekstu i zamknie plik. Sprawdź, czy po ponownym uruchomieniu programu wcześniejsza zawartość pliku została usunięta.

**Zadanie 5.3.3. Dopisywanie nowych danych do pliku**

* Napisz skrypt, który otworzy plik *log.txt* w trybie a i doda do niego nową linię tekstu z aktualną datą i godziną (użyj *datetime.now()* z modułu *datetime*).

**Zadanie 5.3.4. \* Odczyt, modyfikacja i zapis pliku (operacja na każdej linii)**

* Napisz program, który otworzy plik dane.txt w trybie r+, odczyta jego zawartość, a następnie zamieni wszystkie litery na wielkie (upper()) i zapisze zmodyfikowaną zawartość z powrotem do pliku.

**Zadanie 5.3.5. \* Wyszukiwanie i usuwanie linii zawierających określone słowo**

* Napisz skrypt, który otworzy plik *log.txt*, usunie z niego wszystkie linie zawierające słowo "ERROR", a następnie zapisze zmodyfikowaną zawartość do nowego pliku *log\_cleaned.txt.*

## Przetwarzanie danych

Dane są często zapisywane w formatach takich jak CSV (wartości oddzielone przecinkami) oraz JSON (format strukturalny używany w aplikacjach internetowych). Biblioteka pandas umożliwia wygodne operacje na zbiorach danych, takie jak filtrowanie, agregacja czy grupowanie.

Pandas obsługuje CSV, JSON, Excel, Parquet, Pickle, SQL i inne formaty. Biblioteka ułatwia konwersję między formatami i pracę z dużymi zbiorami danych. Można korzystać z różnych trybów zapisu (w, a, wb) i odczytu (r, rb).

Standardowy CSV z separatorem przecinka

|  |
| --- |
| imie,nazwisko,wiek  Anna,Kowalska,25  Jan,Nowak,30 |

CSV z separatorem średnika

|  |
| --- |
| imie;nazwisko;wiek  Anna;Kowalska;25  Jan;Nowak;30 |

Zapis danych do pliku CSV z różnymi separatorami i kodowaniami

|  |
| --- |
| [module\_5\_lesson\_3\_files.py] |

Biblioteka *pandas* umożliwia łatwe wczytywanie, zapisywanie i manipulowanie danymi w plikach CSV. W porównaniu do modułu *csv*, biblioteka *pandas* oferuje większą wygodę, szczególnie w analizie i przetwarzaniu dużych zbiorów danych.

Obsługa plików CSV za pomocą pandas

|  |
| --- |
| [module\_5\_lesson\_3\_files\_pandas.py] |

Format JSON (JavaScript Object Notation) to popularny format przechowywania danych w postaci strukturalnej (klucz-wartość), używany w API, bazach danych i wymianie danych między aplikacjami.

Zapis danych do pliku JSON

|  |
| --- |
| [module\_5\_lesson\_3\_json\_data.py] |

Plik dane.json po zapisie

|  |
| --- |
| [module\_5\_lesson\_3\_json\_table.py] |

Biblioteka pandas umożliwia wygodne przetwarzanie i analizę dużych zbiorów danych. Dzięki niej można szybko filtrować, grupować i agregować dane, a także wykonywać różne operacje na kolumnach i wierszach.

Operacje na danych za pomocą biblioteki *pandas*

|  |
| --- |
| [module\_5\_lesson\_3\_data.py] |

### Zadania

**Zadanie 5.4.1. Wczytanie danych i ich podgląd**

* Załaduj dane o kwiatach irysa (Iris dataset) z biblioteki sklearn i przekonwertuj je na obiekt pandas.DataFrame. Następnie wyświetl pierwsze 5 wierszy oraz podstawowe informacje o zbiorze danych, takie jak liczba wierszy i kolumn.

**Zadanie 5.4.2. Filtrowanie danych**

* Wybierz tylko te kwiaty, których długość płatka (petal length (cm)) jest większa niż 5 cm i szerokość działki kielicha (sepal width (cm)) wynosi co najmniej 3 cm. Wyświetl wyniki.

**Zadanie 5.4.3. Sortowanie danych**

* Posortuj dane według długości płatka (petal length (cm)) w porządku malejącym, a następnie wyświetl pięć pierwszych rekordów.

**Zadanie 5.4.4. Grupowanie i obliczanie średnich wartości**

* Oblicz średnią długość płatka (petal length (cm)) dla każdego gatunku irysa i wyświetl wynik.

**Zadanie 5.4.5. \* Konwersja między formatami i analiza danych**

* Załaduj dane o irysach, a następnie zapisz je w formacie JSON i ponownie wczytaj do pandas. Po ponownym wczytaniu oblicz medianę szerokości działki kielicha (sepal width (cm)) dla każdego gatunku i wyświetl wynik.

## Integracja z zewnętrznymi urządzeniami

Język Python pozwala na wykrywanie podłączonych nośników danych (np. pendrive’ów, dysków zewnętrznych) oraz odczyt i zapis plików na tych urządzeniach. Możliwe jest także programowe sterowanie myszką oraz obsługa kamer. Do przechwytywania obrazu i jego przetwarzania stosuje się bibliotekę *OpenCV*, która pozwala na wykrywanie obiektów, stosowanie filtrów czy analizę obrazu w czasie rzeczywistym.

|  |  |
| --- | --- |
|  | Pamiętaj o instalacji pakietów, m.in. opencv-python, psutil oraz PyAutoGUI. |

### Programowa obsługa myszki

Programowa kontrola myszki pozwala na automatyzację działań wykonywanych przez użytkownika w interfejsie graficznym. Możliwe jest m.in. sterowanie kursorem, klikanie w określone punkty ekranu, przeciąganie elementów oraz symulowanie wpisywania tekstu. Do tego celu wykorzystuje się bibliotekę *pyautogui*, która umożliwia interakcję z systemem operacyjnym bez potrzeby użycia fizycznej myszki czy klawiatury.

Przesunięcie wskaźnika w wybrane miejsce

|  |
| --- |
| [module\_5\_lesson\_4\_mouse.py] |

### Zarządzanie nośnikami

Systemy operacyjne obsługują wiele rodzajów nośników danych, takich jak dyski twarde, pendrive’y, karty pamięci czy napędy sieciowe. Język Python umożliwia programowe wykrywanie podłączonych nośników oraz wykonywanie operacji na ich plikach. Do tego celu można wykorzystać bibliotekę *psutil*, która pozwala pobrać listę dostępnych urządzeń i sprawdzić ich typ oraz system plików.

Wylistowanie dostępnych nośników danych

|  |
| --- |
| [module\_5\_lesson\_4\_disk.py] |

### Programowa obsługa kamery (wymaga 🎥)

Kamery internetowe oraz inne urządzenia rejestrujące obraz mogą być wykorzystywane do przechwytywania i analizy wideo w czasie rzeczywistym. W języku Python biblioteka OpenCV (cv2) umożliwia dostęp do strumienia wideo, podgląd obrazu oraz jego przetwarzanie, np. konwersję do skali szarości, wykrywanie krawędzi czy rozpoznawanie twarzy.

Programowa obsługa kamery

|  |
| --- |
| [module\_5\_lesson\_4\_camera.py] |

### Zadanie

**Zadanie 5.5.1. Wypisanie miejsca, w którym jest kursor**

* Napisz program, który wypisze aktualne położenie kursora. Do tego celu wykorzystaj bibliotekę *pyautogui*.

**Zadanie 5.5.2. Sprawdzenie rozmiaru dysku**

* Napisz skrypt, który pobiera listę dostępnych nośników danych i dla każdego z nich sprawdza ich rozmiar.

**Zadanie 5.5.3. Zapis obrazu z kamery do pliku**

* Napisz program, który przechwyci obraz z kamery i zapisze go do pliku screenshot.png.

**Zadanie 5.5.4. Lista kamer**

* Wypisz listę dostępnych kamer.

**Zadanie 5.5.5. \* Konwersja i odbicie obrazu z kamery na czarno-biały w czasie rzeczywistym**

* Zmodyfikuj kod przechwytujący obraz z kamery tak, aby obraz był wyświetlany w skali szarości oraz odbity w pionie i poziomie. Program powinien działać w pętli i zakończyć działanie po naciśnięciu klawisza q.

## Podsumowanie

W ramach modułu „Klasy, obsługa plików, operacje wejścia/wyjścia, integracja z zewnętrznymi urządzeniami” nauczyliśmy się podstaw programowania obiektowego w języku Python, w tym definiowania i używania klas, obiektów, metod oraz atrybutów. Zrozumieliśmy również, jak obsługiwać popularne błędy pojawiające się podczas wprowadzania danych i pracy z plikami, a także jak tworzyć własne klasy wyjątków, co pozwala na lepszą kontrolę nad błędami w kodzie.

Ponadto zdobyliśmy umiejętność pracy z plikami – otwierania, odczytywania i zapisywania danych w różnych trybach. Poznaliśmy metody przetwarzania danych w formatach takich jak CSV i JSON, korzystając z biblioteki *pandas*. Nauczyliśmy się także integrować program z zewnętrznymi urządzeniami, takimi jak dyski i kamery, co obejmowało programową obsługę myszy oraz przechwytywanie i przetwarzanie obrazu za pomocą biblioteki OpenCV.

## Zadania podsumowujące moduł

**Zadanie 5.7.1. Stwórz klasę Samochod**

* Napisz klasę Samochod, która będzie posiadała dwa atrybuty: marka oraz model. Utwórz obiekt tej klasy i wypisz jego atrybuty na ekran.

**Zadanie 5.7.2. Dodanie metody do klasy**

* Rozszerz klasę Samochod, dodając metodę przedstaw\_sie(), która zwróci tekst w formacie "Samochód marki X, model Y". Utwórz obiekt tej klasy i wywołaj tę metodę.

**Zadanie 5.7.3. Inicjalizacja z wartościami domyślnymi**

* Zmodyfikuj klasę Samochod, tak aby domyślnie przyjmowała markę "Nieznana" i model "Nieznany", jeśli użytkownik nie poda wartości.

**Zadanie 5.7.4. Dziedziczenie – klasa ElektrycznySamochod**

* Utwórz klasę ElektrycznySamochod, która dziedziczy po klasie Samochod i dodaje nowy atrybut pojemnosc\_baterii. Ustaw domyślną wartość na 50 kWh.

**Zadanie 5.7.5. Przesłanianie metod**

* W klasie ElektrycznySamochod nadpisz metodę przedstaw\_sie(), aby dodatkowo informowała o pojemności baterii. Stwórz obiekt tej klasy i wywołaj metodę.

**Zadanie 5.7.6. Tworzenie wielu obiektów**

* Stwórz trzy różne obiekty klasy Samochod, przypisz im różne wartości marka i model, a następnie wypisz ich informacje.

**Zadanie 5.7.7. Liczenie liczby utworzonych obiektów**

* Dodaj do klasy Samochod atrybut klasy liczba\_samochodow, który będzie zwiększał swoją wartość przy każdej nowej instancji. Po utworzeniu kilku obiektów wypisz wartość tego atrybutu.

**Zadanie 5.7.8. Obsługa błędu konwersji danych**

* Napisz program, który pobiera od użytkownika listę liczb oddzielonych przecinkami (np. "3, 5, 7, x, 9") i zamienia je na liczby całkowite. Jeśli jedna z wartości nie jest liczbą, obsłuż wyjątek i pomiń błędne elementy.

**Zadanie 5.7.9. \* Walidacja hasła użytkownika**

* Napisz program, który pobiera od użytkownika hasło i sprawdza, czy spełnia następujące kryteria:
  + Minimum 8 znaków,
  + Co najmniej jedną dużą literę,
  + Co najmniej jedną małą literę,
  + Co najmniej jedną cyfrę,
  + Co najmniej jeden znak specjalny (np. !, @, #, itp.).
* Jeśli hasło nie spełnia któregokolwiek z powyższych kryteriów, zgłoś własny wyjątek, np. InvalidPasswordError, z odpowiednim komunikatem informującym użytkownika, który warunek został niespełniony. Jeśli hasło jest poprawne, wyświetl komunikat "Hasło zaakceptowane!"

**Zadanie 5.7.10. Własny wyjątek dla przekroczenia zakresu**

* Zdefiniuj własny wyjątek PrzekroczonoZakresError, który zostanie zgłoszony, jeśli użytkownik poda liczbę spoza zakresu 1-100.

**Zadanie 5.7.11. Odczyt pliku od końca do początku**

* Napisz program, który utworzy plik tekst.txt z przykładową zawartością, a następnie wypisze jego zawartość od ostatniej do pierwszej linii.

**Zadanie 5.7.12. Zapis listy do pliku w formacie numerowanej listy**

* Napisz program, który pobierze od użytkownika 3 zdań, a następnie zapisze je do pliku lista.txt, numerując każdą linię.

**Zadanie 5.7.13. Usuwanie pustych linii z pliku**

* Napisz program, który wczyta plik tekst.txt, usunie z niego puste linie, a następnie zapisze go ponownie bez pustych wierszy.

**Zadanie 5.7.14. Sprawdzanie czy plik zawiera określone słowo**

* Napisz program, który sprawdzi, czy plik tekst.txt zawiera określone słowo podane przez użytkownika.

**Zadanie 5.7.15. Tworzenie dziennika z datami i wpisami**

* Napisz program, który dopisywać będzie do pliku dziennik.txt nowy wpis z aktualną datą i treścią podaną przez użytkownika.

**Zadanie 5.7.16. Tworzenie i zapis danych do pliku CSV**

* Napisz program, który utworzy plik CSV dane.csv i zapisze do niego kilka wierszy z przykładowymi danymi. Dane powinny zawierać kolumny: imię, nazwisko, wiek.

**Zadanie 5.7.17. Odczyt pliku CSV i obliczenie średniego wieku**

* Napisz program, który odczyta plik dane.csv z poprzedniego zadania i obliczy średni wiek osób.

**Zadanie 5.7.18. Tworzenie i zapis danych do pliku JSON**

* Napisz program, który zapisze słownik danych do pliku dane.json.

**Zadanie 5.7.19. Najstarsza osoba**

* Napisz program, który odczyta plik dane.json i znajdzie najstarszą osobę.

**Zadanie 5.7.20. Dodawania kolumny**

* Napisz program, który wczyta plik dane.csv za pomocą pandas i doda nową kolumnę Rok\_urodzenia.

## Dodatek

**Zadanie 5.1.1.**

|  |
| --- |
| [module\_5\_lesson\_1\_5\_1\_1.py] |

[module\_5\_lesson\_1\_5\_1\_1.png]

**Zadanie 5.1.2.**

|  |
| --- |
| [module\_5\_lesson\_1\_5\_1\_2.py] |

[module\_5\_lesson\_1\_5\_1\_2.png]

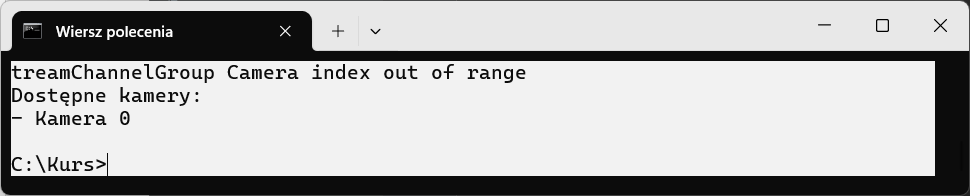
**Zadanie 5.1.3.**

|  |
| --- |
| [module\_5\_lesson\_1\_5\_1\_3.py] |



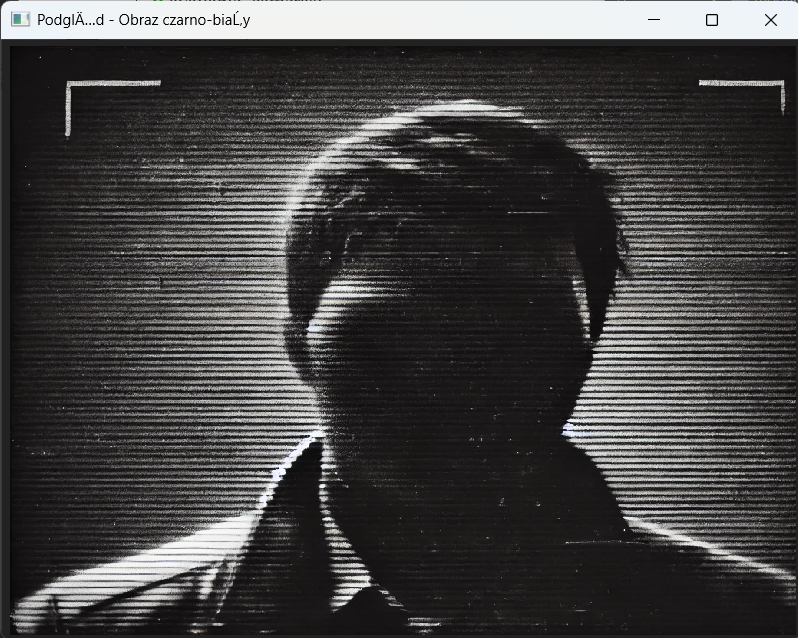
**Zadanie 5.1.4.**

|  |
| --- |
| [module\_5\_lesson\_1\_5\_1\_4.py] |



**Zadanie 5.1.5.**

|  |
| --- |
| [module\_5\_lesson\_1\_5\_1\_5.py] |



**Zadanie 5.2.1.**

|  |
| --- |
| [module\_5\_lesson\_2\_5\_2\_1.py] |

[module\_5\_lesson\_2\_5\_2\_1.png]

**Zadanie 5.2.2.**

|  |
| --- |
| [module\_5\_lesson\_2\_5\_2\_2.py] |

[module\_5\_lesson\_2\_5\_2\_2.png]

**Zadanie 5.2.3.**

|  |
| --- |
| [module\_5\_lesson\_2\_5\_2\_3.py] |

[module\_5\_lesson\_2\_5\_2\_3.png]

**Zadanie 5.2.4.**

|  |
| --- |
| [module\_5\_lesson\_2\_5\_2\_4.py] |

[module\_5\_lesson\_2\_5\_2\_4.png]

**Zadanie 5.2.5.**

|  |
| --- |
| [module\_5\_lesson\_2\_5\_2\_5.py] |

[module\_5\_lesson\_2\_5\_2\_5.png]

**Zadanie 5.3.1.**

|  |
| --- |
| [module\_5\_lesson\_2\_5\_3\_1.py] |

[module\_5\_lesson\_2\_5\_3\_1.png]

**Zadanie 5.3.2.**

|  |
| --- |
| [module\_5\_lesson\_2\_5\_3\_2.py] |

[module\_5\_lesson\_2\_5\_3\_2.png]

**Zadanie 5.3.3.**

|  |
| --- |
| [module\_5\_lesson\_2\_5\_3\_3.py] |

[module\_5\_lesson\_2\_5\_3\_3.png]

**Zadanie 5.3.4.**

|  |
| --- |
| [module\_5\_lesson\_2\_5\_3\_4.py] |

[module\_5\_lesson\_2\_5\_3\_4.png]

**Zadanie 5.3.5.**

|  |
| --- |
| [module\_5\_lesson\_2\_5\_3\_5.py] |

[module\_5\_lesson\_2\_5\_3\_5.png]

**Zadanie 5.4.1.**

|  |
| --- |
| [module\_5\_lesson\_3\_5\_4\_1.py] |

[module\_5\_lesson\_3\_5\_4\_1.png]

**Zadanie 5.4.2.**

|  |
| --- |
| [module\_5\_lesson\_3\_5\_4\_2.py] |

[module\_5\_lesson\_3\_5\_4\_2.png]

**Zadanie 5.4.3.**

|  |
| --- |
| [module\_5\_lesson\_3\_5\_4\_3.py] |

[module\_5\_lesson\_3\_5\_4\_3.png]

**Zadanie 5.4.4.**

|  |
| --- |
| [module\_5\_lesson\_3\_5\_4\_4.py] |

[module\_5\_lesson\_3\_5\_4\_4.png]

**Zadanie 5.4.5.**

|  |
| --- |
| [module\_5\_lesson\_3\_5\_4\_5.py] |

[module\_5\_lesson\_3\_5\_4\_5.png]

**Zadanie 5.5.1.**

|  |
| --- |
| [module\_5\_lesson\_4\_5\_5\_1.py] |

[module\_5\_lesson\_4\_5\_5\_1.png]

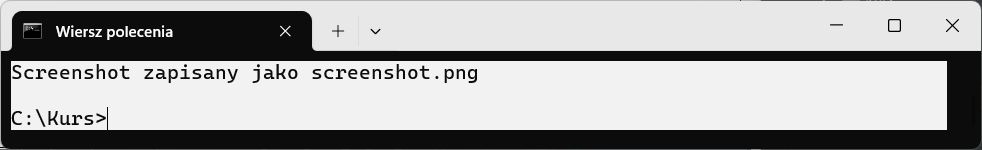
**Zadanie 5.5.2.**

|  |
| --- |
| [module\_5\_lesson\_4\_5\_5\_2.py] |

[module\_5\_lesson\_4\_5\_5\_2.png]

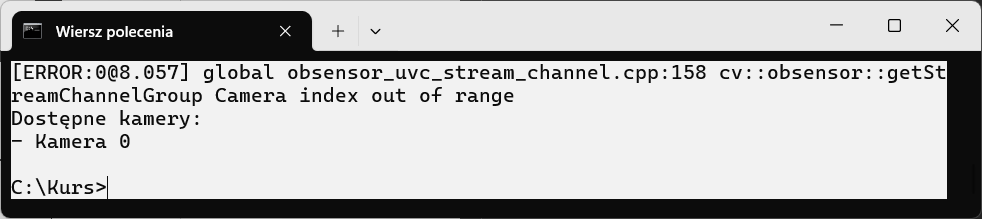
**Zadanie 5.5.3.**

|  |
| --- |
| [module\_5\_lesson\_4\_5\_5\_3.py] |



**Zadanie 5.5.4.**

|  |
| --- |
| [module\_5\_lesson\_4\_5\_5\_4.py] |



**Zadanie 5.5.5.**

|  |
| --- |
| [module\_5\_lesson\_4\_5\_5\_5.py] |



**Zadanie 5.7.1.**

|  |
| --- |
| [module\_5\_summary\_5\_7\_1.py] |

[module\_5\_summary\_5\_7\_1.png]

**Zadanie 5.7.2.**

|  |
| --- |
| [module\_5\_summary\_5\_7\_2.py] |

[module\_5\_summary\_5\_7\_2.png]

**Zadanie 5.7.3.**

|  |
| --- |
| [module\_5\_summary\_5\_7\_3.py] |

[module\_5\_summary\_5\_7\_3.png]

**Zadanie 5.7.4.**

|  |
| --- |
| [module\_5\_summary\_5\_7\_4.py] |

[module\_5\_summary\_5\_7\_4.png]

**Zadanie 5.7.5.**

|  |
| --- |
| [module\_5\_summary\_5\_7\_5.py] |

[module\_5\_summary\_5\_7\_5.png]

**Zadanie 5.7.6.**

|  |
| --- |
| [module\_5\_summary\_5\_7\_6.py] |

[module\_5\_summary\_5\_7\_6.png]

**Zadanie 5.7.7.**

|  |
| --- |
| [module\_5\_summary\_5\_7\_7.py] |

[module\_5\_summary\_5\_7\_7.png]

**Zadanie 5.7.8.**

|  |
| --- |
| [module\_5\_summary\_5\_7\_8.py] |

[module\_5\_summary\_5\_7\_8.png]

**Zadanie 5.7.9.**

|  |
| --- |
| [module\_5\_summary\_5\_7\_9.py] |

[module\_5\_summary\_5\_7\_9.png]

**Zadanie 5.7.10.**

|  |
| --- |
| [module\_5\_summary\_5\_7\_10.py] |

[module\_5\_summary\_5\_7\_10.png]

**Zadanie 5.7.11.**

|  |
| --- |
| [module\_5\_summary\_5\_7\_11.py] |

[module\_5\_summary\_5\_7\_11.png]

**Zadanie 5.7.12.**

|  |
| --- |
| [module\_5\_summary\_5\_7\_12.py] |

[module\_5\_summary\_5\_7\_12.png]

**Zadanie 5.7.13.**

|  |
| --- |
| [module\_5\_summary\_5\_7\_13.py] |

[module\_5\_summary\_5\_7\_13.png]

**Zadanie 5.7.14.**

|  |
| --- |
| [module\_5\_summary\_5\_7\_14.py] |

[module\_5\_summary\_5\_7\_14.png]

**Zadanie 5.7.15.**

|  |
| --- |
| [module\_5\_summary\_5\_7\_15.py] |

[module\_5\_summary\_5\_7\_15.png]

**Zadanie 5.7.16.**

|  |
| --- |
| [module\_5\_summary\_5\_7\_16.py] |

[module\_5\_summary\_5\_7\_16.png]

**Zadanie 5.7.17.**

|  |
| --- |
| [module\_5\_summary\_5\_7\_17.py] |

[module\_5\_summary\_5\_7\_17.png]

**Zadanie 5.7.18.**

|  |
| --- |
| [module\_5\_summary\_5\_7\_18.py] |

[module\_5\_summary\_5\_7\_18.png]

**Zadanie 5.7.19.**

|  |
| --- |
| [module\_5\_summary\_5\_7\_19.py] |

[module\_5\_summary\_5\_7\_19.png]

**Zadanie 5.7.20.**

|  |
| --- |
| [module\_5\_summary\_5\_7\_20.py] |

[module\_5\_summary\_5\_7\_20.png]

# Tworzenie graficznych interfejsów użytkownika

Współczesne aplikacje nie tylko przetwarzają dane, ale także prezentują je w sposób czytelny i intuicyjny dla użytkownika. Celem tego modułu jest rozwinięcie umiejętności pozwalających na projektowanie graficznych interfejsów użytkownika (GUI) w języku Python. Nauczymy się, jak tworzyć interaktywne aplikacje biblioteką *Tkinter*, które umożliwią obsługę formularzy i pól wejściowych. Zapoznamy się również z bibliotekami, które pozwalają na wizualizację danych w formie tabel i wykresów.

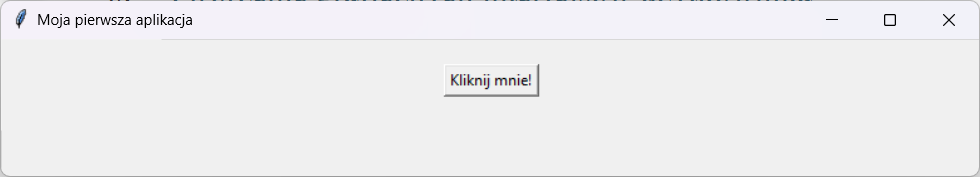
## Wprowadzenie do Tkinter

Podczas lekcji wykorzystamy całkowicie darmową bibliotekę *Tkinter*, która jest popularnym narzędziem do tworzenia GUI. W ramach zajęć omówimy aspekty budowy aplikacji okienkowej – od podstawowych okien dialogowych i przycisków, aż po bardziej zaawansowane interfejsy z formularzami i dynamiczną prezentacją danych.

Język pozwala na łatwe utworzenie głównego okna aplikacji za pomocą biblioteki Tkinter, nadać mu tytuł i dodać interaktywny przycisk. W prezentowanym przypadku, po kliknięciu przycisku zostanie wyświetlona wiadomość w oknie dialogowym.

Tworzenie okna za pomocą biblioteki tkinter

|  |
| --- |
| [module\_6\_lesson\_1\_tk.py] |

Okno wygenerowane biblioteką tkinter

Poniższy kod tworzy prostą aplikację GUI przy użyciu biblioteki Tkinter. Jej działanie przebiega według następujących kroków:

1. Tworzenie głównego okna – Za pomocą *tk.Tk()* inicjalizowane jest główne okno aplikacji, a następnie nadawany jest mu tytuł "Moja pierwsza aplikacja".
2. Definicja funkcji obsługującej zdarzenie – Funkcja *pokaz\_wiadomosc()* zostaje zdefiniowana; po jej wywołaniu wyświetla się okno dialogowe z komunikatem "Witaj w aplikacji!".
3. Tworzenie przycisku – Za pomocą *tk.Button()* tworzony jest przycisk z napisem "Kliknij mnie!", który po kliknięciu wywołuje funkcję *pokaz\_wiadomosc()*. Przycisk jest dodawany do okna i wyśrodkowany poprzez *pack(pady=20)*.
4. Uruchomienie pętli zdarzeń – *root.mainloop()* włącza główną pętlę programu, dzięki której aplikacja reaguje na interakcje użytkownika, np. kliknięcia przycisku.

|  |  |
| --- | --- |
|  | Podobnie jak w przypadku innych bibliotek, plik z kodem nie może mieć nazwy biblioteki 'tkinter.py' - może to doprowadzić do konfliktu nazw plików. |

### Zadania

**Zadanie 6.1.1. Sprawdzenie instalacji Tkinter**

* Napisz skrypt, który sprawdzi, czy biblioteka *Tkinter* jest dostępna w systemie. Użyj import tkinter i obsłuż ewentualny wyjątek.

**Zadanie 6.1.2. czy Tkinter jest zainstalowany**

* Napisz kod, który wyświetli wersję zainstalowanej biblioteki *Tkinter*. Możesz użyć *tk.TkVersion* lub *help('tkinter')*.

**Zadanie 6.1.3. Sprawdzenie obecności pliku tkinter.py**

* Napisz skrypt, który sprawdzi, czy w bieżącym katalogu nie ma pliku o nazwie tkinter.py, ponieważ może on powodować konflikty. Użyj modułu os i os.listdir().

**Zadanie 6.1.4. Dostępne obiekty w Tkinter**

* Napisz program, który wyświetli listę dostępnych funkcji, klas i obiektów w bibliotece Tkinter. Możesz użyć funkcji dir(tkinter).

**Zadanie 6.1.5. \* Wyświetl wersję Tkinter w oknie dialogowym**

* Napisz program, który otworzy okno Tkinter i wyświetli jego wersję w komunikacie. Możesz użyć tk.TkVersion i messagebox.showinfo().

## Tworzenie bardziej zaawansowanego interfejsu – formularz i pola wejściowe

W Tkinter, podstawową metodą wprowadzania danych przez użytkownika są pola wejściowe. Pozwalają one na wpisanie tekstu, liczb lub innych danych, które można później przetworzyć w programie.

Tworzenie prostego formularza z danymi wejściowymi

|  |  |
| --- | --- |
| **Pole** | **Opis** |
| Entry | Pole do wprowadzania krótkich tekstów lub liczb. |
| Text | Pole do wprowadzania wieloliniowego tekstu. |
| Spinbox | Wybór wartości liczbowej z listy. |
| Checkbutton  Radiobutton  Combobox | Interaktywne elementy do wyboru wartości. |

Formularz i funkcja do pobrania i wyświetlenia danych

|  |
| --- |
| [module\_6\_lesson\_2\_tk\_more.py] |

Formularz z przyciskami

Pola wejściowe w Tkinter mogą mieć różne parametry, takie jak szerokość, maksymalna liczba znaków, domyślna wartość czy możliwość edycji. Aby sprawdzić dostępne opcje dla danego typu pola, można użyć funkcji *help()*, np.: *help(tk.Entry)*.

### Zadania

**Zadanie 6.2.1. Tworzenie okna z etykietą i zamykaniem**

* Napisz program, który utworzy okno Tkinter z etykietą "Witaj w aplikacji!" oraz przyciskiem "Zamknij", który zamknie okno po kliknięciu.

**Zadanie 6.2.2. Ustawianie tytułu i rozmiaru okna**

* Zmodyfikuj kod z poprzedniego zadania tak, aby okno miało tytuł „Moja Aplikacja Tkinter” i rozmiar 400x300 pikseli.

**Zadanie 6.2.3. Utworzenie CheckBox**

* Dodaj do okna CheckBox z etykietą „To jest Checkbox”. Po jego zaznaczeniu wyświetl komunikat o jego kliknięciu za pomocą messagebox.showinfo(‘x;,’y’).

**Zadanie 6.2.4. \* Kopiowanie tekstu z jednego pola do drugiego**

* Stwórz aplikację z dwoma polami tekstowymi i przyciskiem. Po kliknięciu przycisku tekst z pierwszego pola ma być skopiowany do drugiego pola.

**Zadanie 6.2.5. \* Licznik kliknięć**

* Napisz program, w którym znajduje się przycisk. Po każdym kliknięciu jego wartość powinna zwiększać się o 1 i być wyświetlana na ekranie.

## Wizualizacja danych – prezentacja tabel i wykresów

Aplikacje często wymagają przechowywania i przetwarzania danych, dlatego ważną umiejętnością jest ich zapisywanie i odczytywanie. W tym rozdziale sprawdzimy, jak wczytywać i zapisywać dane w plikach, oraz prezentować je w przejrzystej formie tabelarycznej. Dodatkowo zostanie omówione , jak tworzyć proste wykresy przy użyciu biblioteki Matplotlib – zaawansowanej i w pełni darmowej biblioteki przeznaczonej do wizualizacji danych.

Gotowe zbiory danych, dostępne w bibliotece *Scikit-learn* pozwalają na szybką eksplorację danych. Jednym z najczęściej używanych zestawów jest zbiór Iris, który zawiera informacje o długości i szerokości działki kielicha oraz płatka dla trzech gatunków irysów. W poniższym kodzie dane te są wczytywane, konwertowane do formatu *DataFrame*, a następnie wzbogacane o kolumnę z oznaczeniem gatunku.

Poniższy kod tworzy zestaw danych o średnich cenach mieszkań w polskich miastach, zapisuje go do obiektu DataFrame, a następnie eksportuje do pliku CSV. Dzięki temu dane mogą być łatwo przechowywane i używane w przyszłych analizach.

Tworzenie i zapis danych do CSV

|  |
| --- |
| [module\_6\_lesson\_3\_dataset.py] |

Następnie, kod wczytuje zapisany wcześniej plik CSV do obiektu *DataFrame* i wyświetla jego zawartość w formie tabeli. Pozwala to na szybki podgląd danych przed ich dalszą analizą lub wizualizacją.

Odczyt danych z pliku i prezentacja w tabeli

|  |
| --- |
| [module\_6\_lesson\_3\_table.py] |

Następnie, utwórzmy wykres słupkowy, gdzie każde miasto reprezentuje słupek, a jego wysokość odpowiada średniej cenie za m². Dodane są także opisy osi i tytuł, co sprawia, że wykres jest czytelny i łatwy do interpretacji.

Wizualizacja danych - wykres słupkowy cen mieszkań

|  |
| --- |
| [module\_6\_lesson\_3\_plot.py] |

Oprócz wykresu słupkowego, biblioteka pozwala tworzyć wykres liniowy, który świetnie nadaje się do analizy zmian cen mieszkań w czasie, np. w poszczególnych latach. Wykres kołowy może być użyty do przedstawienia procentowego udziału miast w ogólnej sumie cen mieszkań - pozwala to zobaczyć, które miasto ma najwyższe ceny. Histogram może prezentować rozkład cen mieszkań, pomagając zrozumieć, czy ceny są równomiernie rozłożone, czy może istnieją przedziały cenowe, w których mieszkania są bardziej popularne.

### Zadania

**Zadanie 6.3.1. Biblioteki matplotlib, sklearn oraz pandas**

* Napisz program, który zaimportuje biblioteki matplotlib, sklearn i pandas oraz sprawdzi ich wersje. Wypisz wersje tych bibliotek w konsoli.

**Zadanie 6.3.2. Wczytanie i wyświetlenie danych**

* Wczytaj dane z pliku ceny\_mieszkan.csv i wyświetl pierwsze 5 wierszy w tabeli. Użyj biblioteki pandas.

**Zadanie 6.3.3. Wykres słupkowy cen mieszkań**

* Narysuj wykres słupkowy przedstawiający średnie ceny mieszkań w różnych miastach, korzystając z danych z pliku ceny\_mieszkan.csv. Oznacz osie oraz dodaj tytuł wykresu.

**Zadanie 6.3.4. \* Wykres liniowy zmian cen**

* Stwórz wykres liniowy pokazujący zmiany średnich cen mieszkań w jednym z miast w różnych latach. Wykorzystaj przykładowe dane historyczne zapisane w DataFrame.

**Zadanie 6.3.5. \* Analiza korelacji cen mieszkań**

* Wczytaj dane o cenach mieszkań i wykorzystaj sklearn do obliczenia korelacji między cenami mieszkań w różnych miastach. Zinterpretuj wynik korelacji i wyświetl go w konsoli.

## Podsumowanie

W tym module nauczyłeś się, jak tworzyć graficzne interfejsy użytkownika w języku Python przy użyciu biblioteki Tkinter. Poznałeś podstawowe komponenty GUI, takie jak etykiety, przyciski oraz pola wejściowe, a także bardziej zaawansowane elementy, np. obiekty typu *CheckBox* czy dynamiczne formularze. Pracowałeś również z danymi, ucząc się, jak je przechowywać, wczytywać oraz wizualizować w formie tabel i wykresów. Dzięki wykorzystaniu bibliotek takich jak *Pandas*, *Matplotlib* i *Scikit-learn*, możesz analizować dane i prezentować je w przejrzysty sposób, co jest nieocenione przy budowie profesjonalnych aplikacji

## Zadania podsumowujące moduł

**Zadanie 6.5.1. Tworzenie okna Tkinter ze swoim imieniem w tytule**

* Napisz program, który utworzy podstawowe okno Tkinter z tytułem i określonym rozmiarem 1000x300

**Zadanie 6.5.2. Dodanie przycisku do okna**

Dodaj przycisk do okna Tkinter, który po kliknięciu wyświetli komunikat „Witaj!”.

**Zadanie 6.5.3. Tworzenie formularza**

* Stwórz formularz z polami do wprowadzania imienia, nazwiska i wieku, a następnie dodaj przycisk, który wyświetli wpisane dane.

**Zadanie 6.5.4. Obsługa CheckBox**

* Dodaj do okna Tkinter CheckBox z opcją „Akceptuję regulamin” i sprawdź, czy użytkownik go zaznaczył.

**Zadanie 6.5.5. Obsługa pola wyboru (Combobox)**

* Utwórz listę rozwijaną z wyborem miast i wyświetl wybrane miasto po naciśnięciu przycisku.

**Zadanie 6.5.6. Obsługa pola tekstowego (odwracanie tekstu)**

* Napisz program, który pozwoli na wpisanie tekstu w polu Entry i odwrócenie go po kliknięciu przycisku.

**Zadanie 6.5.7. Mnożenie po kliknięciu**

* Stwórz program, który będzie mnożył wpisaną liczbę przez 2 po kliknięciu przycisku.

**Zadanie 6.5.8. Wczytanie danych z pliku CSV**

* Wczytaj dane z pliku ceny\_mieszkan.csv i wyświetl pierwsze 10 wierszy w tabeli.

**Zadanie 6.5.9. Zapis Dataset z ponumerowanymi dniami tygodnia do pliku CSV**

* Stwórz prosty DataFrame i zapisz go do pliku CSV.

**Zadanie 6.5.10. Tworzenie wykresu słupkowego dni tygodnia**

* Narysuj wykres słupkowy przedstawiający dni tygodnia z liczbami.

**Zadanie 6.5.11. Tworzenie wykresu kołowego dni tygodnia**

* Narysuj wykres kołowy pokazujący dni tygodnia z liczbami.

**Zadanie 6.5.12. Tworzenie wykresu liniowego**

* Narysuj wykres liniowy przedstawiający zmiany cen mieszkań w jednym mieście na przestrzeni lat.

**Zadanie 6.5.13. Histogram cen mieszkań**

* Stwórz histogram pokazujący rozkład cen mieszkań w różnych przedziałach cenowych.

**Zadanie 6.5.14. Interaktywna aplikacja Tkinter z wykresem**

* Stwórz aplikację Tkinter, która pozwoli użytkownikowi wybrać miasto, a następnie wyświetli wykres cen mieszkań dla tego miasta.

**Zadanie 6.5.15. Sortowanie danych w tabeli**

* Wczytaj dane o cenach mieszkań i posortuj je według ceny za m² w kolejności rosnącej.

**Zadanie 6.5.16. Obliczanie średniej ceny mieszkań**

* Oblicz średnią cenę mieszkań na podstawie danych z pliku CSV.

**Zadanie 6.5.17. Filtracja danych**

* Wczytaj dane o cenach mieszkań i wyświetl tylko te miasta, w których cena za m² jest wyższa niż 10 000 PLN.

**Zadanie 6.5.18. Analiza korelacji cen mieszkań**

* Oblicz korelację między cenami mieszkań w różnych miastach i przedstaw wynik w formie tabeli.

**Zadanie 6.5.19. Tworzenie dynamicznych formularzy**

* Stwórz aplikację Tkinter, która pozwala użytkownikowi dynamicznie dodawać pola do formularza.

**Zadanie 6.5.20. Tworzenie aplikacji końcowej**

* Stwórz pełną aplikację w Tkinter, która pozwoli na wczytanie danych z pliku CSV, wyświetlenie ich w tabeli i wygenerowanie wykresów na podstawie tych danych.

## Dodatek

**Zadanie 6.1.1.**

|  |
| --- |
| [module\_6\_lesson\_1\_6\_1\_1.py] |

[module\_6\_lesson\_1\_6\_1\_1.png]

**Zadanie 6.1.2.**

|  |
| --- |
| [module\_6\_lesson\_1\_6\_1\_2.py] |

[module\_6\_lesson\_1\_6\_1\_2.png]

**Zadanie 6.1.3.**

|  |
| --- |
| [module\_6\_lesson\_1\_6\_1\_3.py] |

[module\_6\_lesson\_1\_6\_1\_3.png]

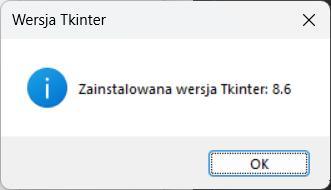
**Zadanie 6.1.4.**

|  |
| --- |
| [module\_6\_lesson\_1\_6\_1\_4.py] |

[module\_6\_lesson\_1\_6\_1\_4.png]

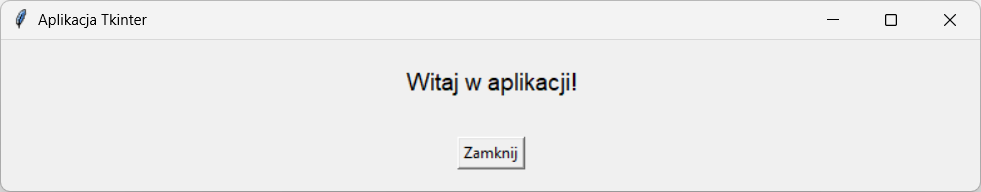
**Zadanie 6.1.5.**

|  |
| --- |
| [module\_6\_lesson\_1\_6\_1\_5.py] |



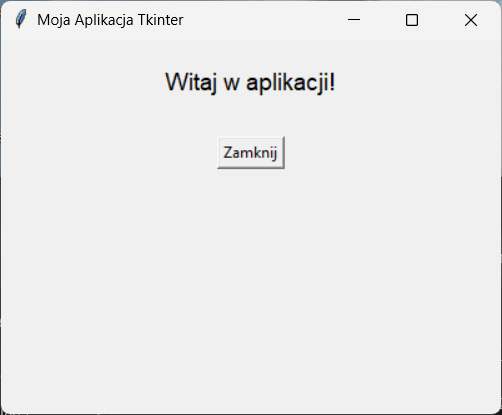
**Zadanie 6.2.1.**

|  |
| --- |
| [module\_6\_lesson\_2\_6\_2\_1.py] |



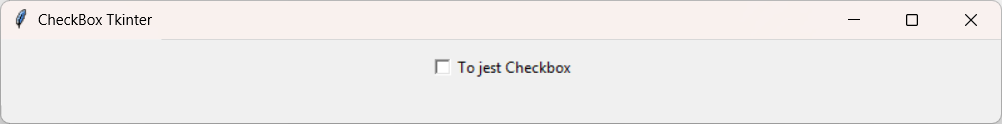
**Zadanie 6.2.2.**

|  |
| --- |
| [module\_6\_lesson\_2\_6\_2\_2.py] |



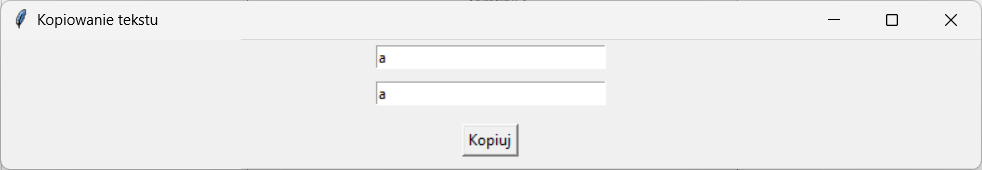
**Zadanie 6.2.3.**

|  |
| --- |
| [module\_6\_lesson\_2\_6\_2\_3.py] |



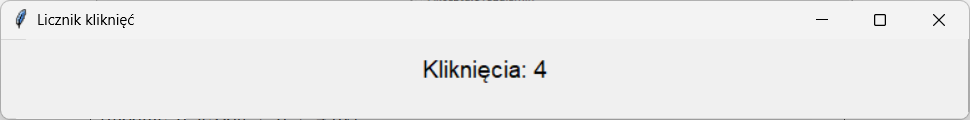
**Zadanie 6.2.4.**

|  |
| --- |
| [module\_6\_lesson\_2\_6\_2\_4.py] |



**Zadanie 6.2.5.**

|  |
| --- |
| [module\_6\_lesson\_2\_6\_2\_5.py] |



**Zadanie 6.3.1.**

|  |
| --- |
| [module\_6\_lesson\_3\_6\_3\_1.py] |

[module\_6\_lesson\_3\_6\_3\_1.png]

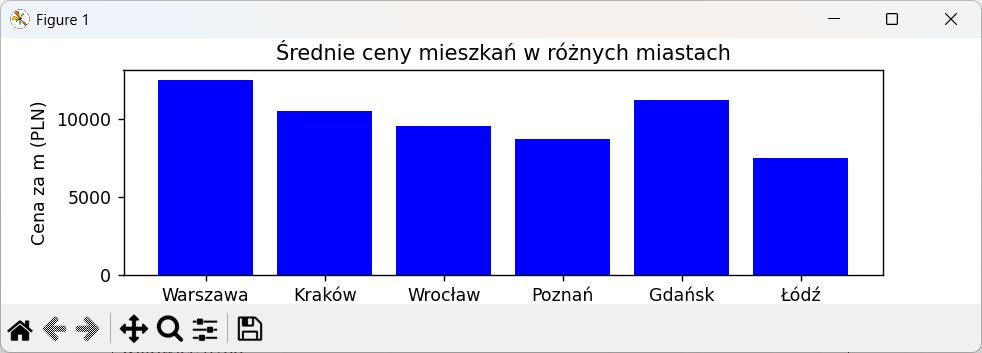
**Zadanie 6.3.2.**

|  |
| --- |
| [module\_6\_lesson\_3\_6\_3\_2.py] |

[module\_6\_lesson\_3\_6\_3\_2.png]

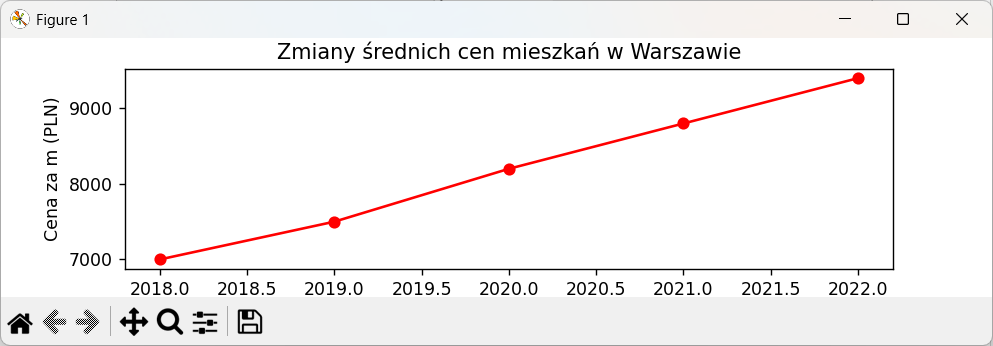
**Zadanie 6.3.3.**

|  |
| --- |
| [module\_6\_lesson\_3\_6\_3\_3.py] |



**Zadanie 6.3.4.**

|  |
| --- |
| [module\_6\_lesson\_3\_6\_3\_4.py] |



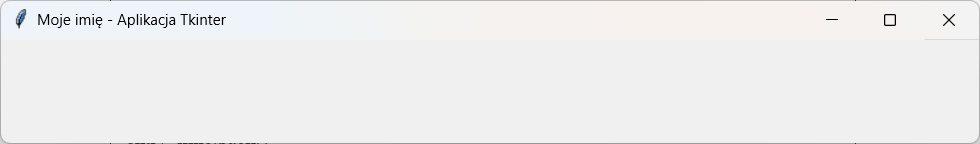
**Zadanie 6.3.5.**

|  |
| --- |
| [module\_6\_lesson\_3\_6\_3\_5.py] |

[module\_6\_lesson\_3\_6\_3\_5.png]

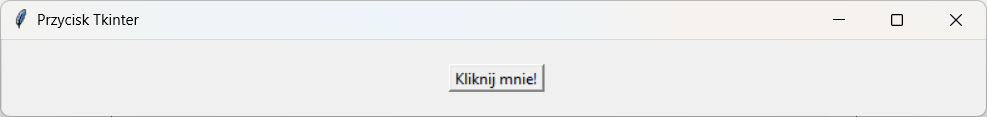
**Zadanie 6.5.1.**

|  |
| --- |
| [module\_6\_summary\_6\_5\_1.py] |



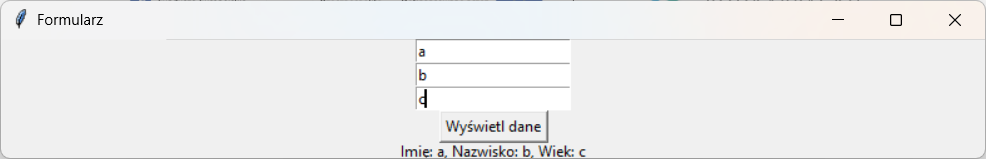
**Zadanie 6.5.2.**

|  |
| --- |
| [module\_6\_summary\_6\_5\_2.py] |



**Zadanie 6.5.3.**

|  |
| --- |
| [module\_6\_summary\_6\_5\_3.py] |



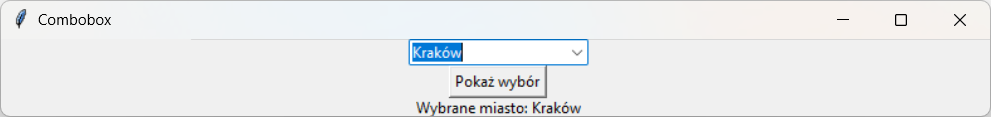
**Zadanie 6.5.4.**

|  |
| --- |
| [module\_6\_summary\_6\_5\_4.py] |



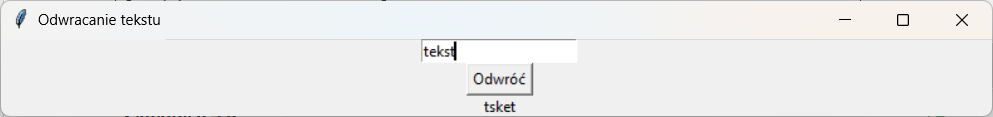
**Zadanie 6.5.5.**

|  |
| --- |
| [module\_6\_summary\_6\_5\_5.py] |



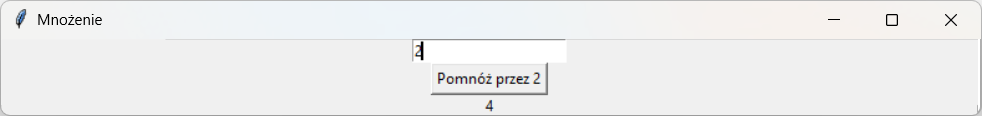
**Zadanie 6.5.6.**

|  |
| --- |
| [module\_6\_summary\_6\_5\_6.py] |



**Zadanie 6.5.7.**

|  |
| --- |
| [module\_6\_summary\_6\_5\_7.py] |



**Zadanie 6.5.8.**

|  |
| --- |
| [module\_6\_summary\_6\_5\_8.py] |

[module\_6\_summary\_6\_5\_8.png]

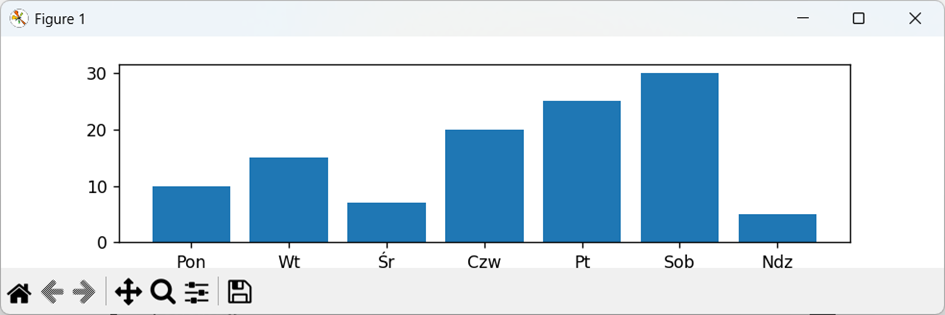
**Zadanie 6.5.9.**

|  |
| --- |
| [module\_6\_summary\_6\_5\_9.py] |

[module\_6\_summary\_6\_5\_9.png]

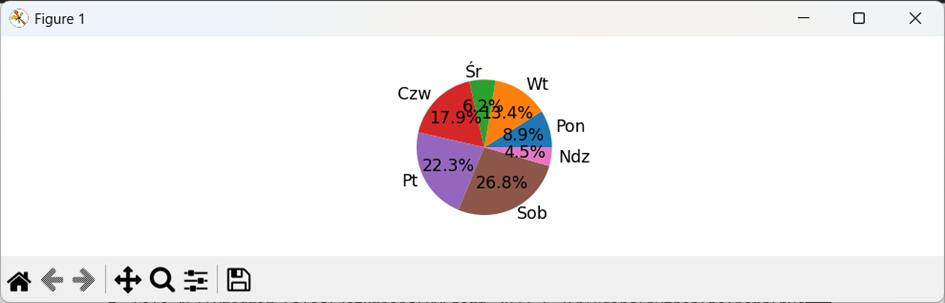
**Zadanie 6.5.10.**

|  |
| --- |
| [module\_6\_summary\_6\_5\_10.py] |



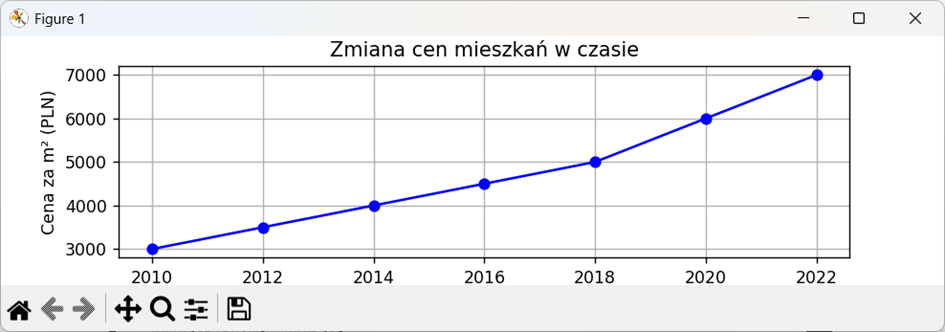
**Zadanie 6.5.11.**

|  |
| --- |
| [module\_6\_summary\_6\_5\_11.py] |



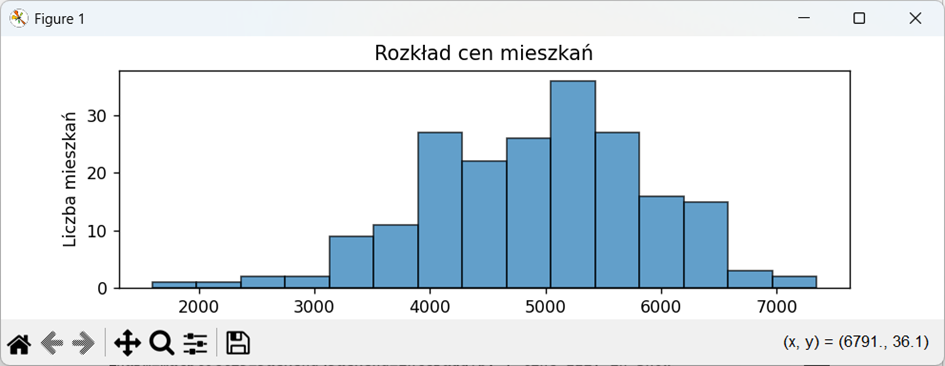
**Zadanie 6.5.12.**

|  |
| --- |
| [module\_6\_summary\_6\_5\_12.py] |



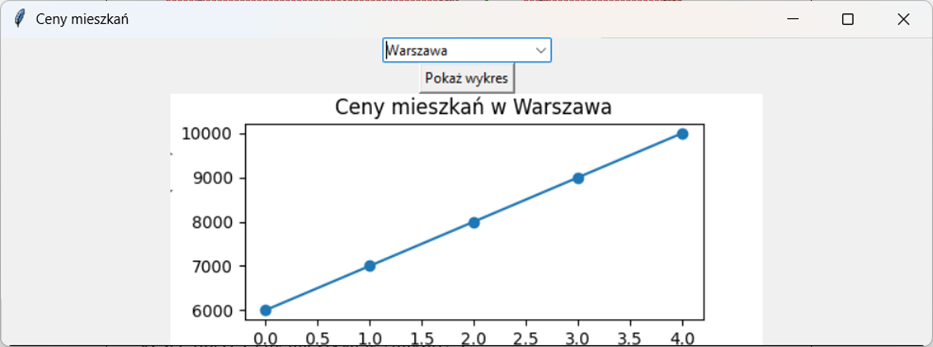
**Zadanie 6.5.13.**

|  |
| --- |
| [module\_6\_summary\_6\_5\_13.py] |



**Zadanie 6.5.14.**

|  |
| --- |
| [module\_6\_summary\_6\_5\_14.py] |



**Zadanie 6.5.15.**

|  |
| --- |
| [module\_6\_summary\_6\_5\_15.py] |

[module\_6\_summary\_6\_5\_15.png]

**Zadanie 6.5.16.**

|  |
| --- |
| [module\_6\_summary\_6\_5\_16.py] |

[module\_6\_summary\_6\_5\_16.png]

**Zadanie 6.5.17.**

|  |
| --- |
| [module\_6\_summary\_6\_5\_17.py] |

[module\_6\_summary\_6\_5\_17.png]

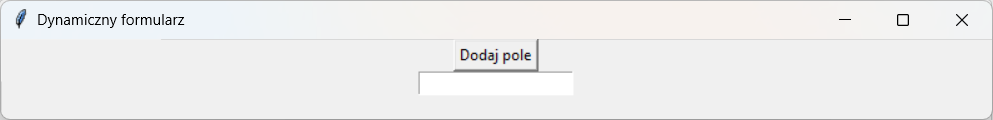
**Zadanie 6.5.18.**

|  |
| --- |
| [module\_6\_summary\_6\_5\_18.py] |

[module\_6\_summary\_6\_5\_18.png]

**Zadanie 6.5.19.**

|  |
| --- |
| [module\_6\_summary\_6\_5\_19.py] |

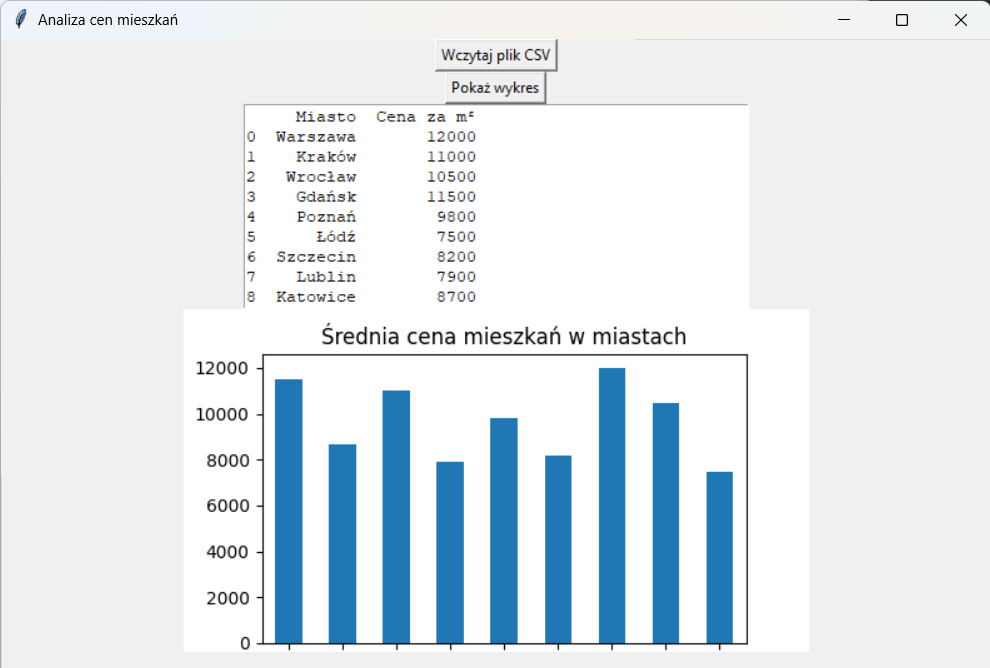


**Zadanie 6.5.20.**

|  |
| --- |
| [module\_6\_summary\_6\_5\_20.py] |

Dane wejściowe CSV

|  |  |
| --- | --- |
| |  | | --- | | Miasto,Cena za m²  Warszawa,12000  Kraków,11000  Wrocław,10500  Gdańsk,11500  Poznań,9800  Łódź,7500  Szczecin,8200  Lublin,7900  Katowice,8700 | |



1. Strona projektu Python - https://www.python.org/ [↑](#footnote-ref-2)
2. PyPI · The Python Package Index - https://pypi.org/ [↑](#footnote-ref-3)
3. Strona do pobrania Thonny IDE - https://thonny.org/ [↑](#footnote-ref-4)
4. Strona do pobrania Visual Studio Code - [↑](#footnote-ref-5)
5. https://peps.python.org/pep-0008/ [↑](#footnote-ref-6)