Nauka programowania w językach Scratch i Python

Python – poziom podstawowy

Materiał dla nauczycieli

Spis treści

[1. Podstawy języka Python 2](#_Toc1675953195)

[1.1. Język Python, jego rola i podstawowe narzędzia 3](#_Toc590910770)

[1.1.1. Wprowadzenie 3](#_Toc203577016)

[1.1.1.1. Struktura kursu i sposób nauki 3](#_Toc1131110189)

[1.1.1.2. Zastosowanie języka 3](#_Toc1331285428)

[1.1.1.3. Wady i zalety języka Python 4](#_Toc991851563)

[1.1.1.4. Wykorzystanie języka Python – interaktywnie czy przez skrypt? 4](#_Toc1505514581)

[1.1.2. Algorytmika – czym jest? 5](#_Toc1053182221)

[1.1.3. Podstawowe narzędzia kursu 6](#_Toc1107410087)

[1.1.3.1. Systemy operacyjne (Windows oraz Ubuntu) 7](#_Toc109786928)

[1.1.3.2. Środowiska i edytory (Thonny, Visual Studio Code) 7](#_Toc1256810630)

[1.1.3.3. Uruchamianie kodu języka Python w terminalu 8](#_Toc823198706)

[1.1.4. Zadania 10](#_Toc1518623514)

[1.2. Instalacja środowiska Python w różnych systemach operacyjnych 10](#_Toc974597472)

[1.2.1. Instalacja środowiska Python na systemie Linux 11](#_Toc103207862)

[1.2.2. Instalacja środowiska Python na systemie Windows 16](#_Toc586718153)

[1.2.3. Weryfikacja wersji interpretera języka Python 19](#_Toc663980391)

[1.2.4. Tworzenie wirtualnych środowisk 20](#_Toc1558845136)

[1.2.5. Zarządzanie wersjami 21](#_Toc2039261131)

[1.2.6. Zadania 22](#_Toc1630224163)

[1.3. Zasady tworzenia i struktury skryptów w języku Python 23](#_Toc1810595504)

[1.3.1. Struktura plików .py 23](#_Toc724582786)

[1.3.2. Pierwsze skrypty w edytorze tekstowym 24](#_Toc1347346280)

[1.3.3. Uruchamianie skryptów Python w konsoli systemowej 24](#_Toc1612850763)

[1.3.4. Wybór trybu pracy: interaktywny czy skryptowy? 25](#_Toc1119054655)

[1.3.5. Zadania 27](#_Toc262325410)

[1.4. Uruchamianie kodu w konsoli za pomocą interpretera 28](#_Toc421678750)

[1.4.1. Praca z interaktywną konsolą Python 29](#_Toc1789394721)

[1.4.2. Wprowadzanie pojedynczych komend i natychmiastowe obserwowanie wyników 30](#_Toc537594037)

[1.4.3. Testowanie fragmentów kodu w trybie interaktywnym 30](#_Toc1678264022)

[1.4.4. Zadania 32](#_Toc310091147)

[1.5. Prowadzenie operacji matematycznych i manipulacji tekstowych 32](#_Toc673815994)

[1.5.1. Podstawowe operacje arytmetyczne w języku Python 33](#_Toc141164781)

[1.5.2. Manipulacja napisami 34](#_Toc2108182119)

[1.5.3. Wykorzystanie wbudowanych funkcji: len() i type() 36](#_Toc2039087644)

[1.5.4. Zadania 36](#_Toc372079117)

[1.6. Typy zmiennych języka Python 37](#_Toc930194341)

[1.6.1. Typy liczbowe 38](#_Toc668281283)

[1.6.2. Typy sekwencyjne 41](#_Toc881509953)

[1.6.3. Zbiory 43](#_Toc255583226)

[1.6.4. Typ mapujący 44](#_Toc515764164)

[1.6.5. Typ braku wartości 45](#_Toc528903194)

[1.6.6. Zadania 45](#_Toc72910743)

[1.7. Używanie metod wejścia i wyjścia 46](#_Toc2055368942)

[1.7.1. Funkcja input() 46](#_Toc1697186953)

[1.7.2. Funkcja print() 47](#_Toc336401615)

[1.7.3. Formatowanie stringów 48](#_Toc1501483765)

[1.7.4. Zadania 48](#_Toc46651993)

[1.8. Korzystanie z bibliotek i menadżera PIP 49](#_Toc241759291)

[1.8.1. Instalacja i wykorzystanie bibliotek z pomocą narzędzia PIP 50](#_Toc1822208333)

[1.8.2. Instalacja i korzystanie z bibliotek na przykładzie turtle 51](#_Toc871011417)

[1.8.3. Zadania 53](#_Toc1220565172)

[1.9. Podsumowanie 53](#_Toc354027398)

[1.10. Zadania podsumowujące 54](#_Toc2081740872)

[1.11. Dodatek 57](#_Toc465512029)

[2. Rozpoczynamy kodowanie w języku Python 67](#_Toc1660252841)

[2.1. Przygotowanie środowiska IDE 68](#_Toc1794751872)

[2.1.1. Omówienie różnych IDE 68](#_Toc771999448)

[2.1.2. Instalacja wybranych IDE: Thonny i Visual Studio Code 70](#_Toc952695949)

[2.1.3. Zapoznanie się z funkcjami Thonny 71](#_Toc58719629)

[2.1.4. Zapoznanie się z funkcjami VS Code 72](#_Toc929982530)

[2.1.5. Zadania 73](#_Toc96650217)

[2.2. Zrozumienie zasad wcięć oraz formatowania kodu 73](#_Toc1271184499)

[2.2.1. Zasady PEP 8 74](#_Toc2053069454)

[2.2.2. Rola wcięć w kodzie Python 74](#_Toc704101381)

[2.2.3. Zadania 74](#_Toc317080107)

[2.3. Pierwszy program w IDE 75](#_Toc828134304)

[2.3.1. Przykład napisania kodu 75](#_Toc1316793130)

[2.3.2. Analiza działania kodu i zwracanych błędów 75](#_Toc2066042447)

[2.3.3. Zadania 76](#_Toc1087201918)

[2.4. Zmienne i komentarze 77](#_Toc87133993)

[2.4.1. Deklarowanie zmiennych w Pythonie 77](#_Toc1944019999)

[2.4.2. Wartość zmiennej i jej typ 78](#_Toc130025874)

[2.4.3. Komentarze w kodzie 78](#_Toc1321937295)

[2.4.4. Zadania 79](#_Toc525662563)

[2.5. Podsumowanie 79](#_Toc1268818753)

[2.6. Zadania 80](#_Toc1753508594)

[2.7. Dodatek 82](#_Toc1437421215)

[3. Logika warunkowa i kontrola przepływu programu w kodzie źródłowym 88](#_Toc228944714)

[3.1. Wprowadzenie do logiki warunkowej 88](#_Toc2146422977)

[3.1.1. Zadania 88](#_Toc1336949338)

[3.2. Instrukcja if oraz budowa złożonych wyrażeń 89](#_Toc418177930)

[3.2.1. Zadania 90](#_Toc1105925251)

[3.3. Logika warunkowa w praktyce 90](#_Toc622709923)

[3.3.1. Zadania 91](#_Toc2045444539)

[3.4. Konwersje typów 91](#_Toc1763080292)

[3.4.1. Zadania 92](#_Toc246507971)

[3.5. Podsumowanie 92](#_Toc688238081)

[3.6. Zadania 93](#_Toc7038490)

[3.7. Dodatek 95](#_Toc1695428342)

[4. Napisy i działania matematyczne 99](#_Toc149281415)

[4.1. Tworzenie napisów i działanie na łańcuchach znaków 100](#_Toc836793794)

[4.1.1. Zadania 100](#_Toc1097457953)

[4.2. Działania matematyczne i liczby 101](#_Toc449792452)

[4.2.1. Zadania 101](#_Toc618231070)

[4.3. Zapoznanie z obiektami: Listy, krotki i słowniki 101](#_Toc803857302)

[4.3.1. Zadania 102](#_Toc6727732)

[4.4. Podsumowanie 103](#_Toc1919610951)

[4.5. Zadania 103](#_Toc2041970650)

[4.6. Dodatek 105](#_Toc791925993)

[5. Funkcje i pętle 110](#_Toc105926560)

[5.1. Tworzenie funkcji w języku Python 111](#_Toc1640408278)

[5.1.1. Tworzenie prostych funkcji 111](#_Toc566952442)

[5.1.2. Dokumentowanie funkcji za pomocą docstring’ów 112](#_Toc1203919902)

[5.1.3. Zadania 112](#_Toc755399890)

[5.2. Zrozumienie i zastosowanie pętli 113](#_Toc1556382582)

[5.2.1. Składnia i działanie pętli for i while 113](#_Toc1476954760)

[5.2.2. Zadania iteracyjne z zastosowaniem funkcji 114](#_Toc99467501)

[5.2.3. Zadania 115](#_Toc954915319)

[5.3. Rekurencja i zasięg zmiennych 116](#_Toc1908534310)

[5.3.1. Zrozumienie rekurencji i tworzenia funkcji rekurencyjnych 116](#_Toc1939081196)

[5.3.2. Zmienne globalne a zmienne lokalne 116](#_Toc484915836)

[5.3.3. Zadania 117](#_Toc162874482)

[5.4. Podsumowanie 118](#_Toc1884726073)

[5.5. Zadania 118](#_Toc1234478754)

[5.6. Dodatek 119](#_Toc498680465)

[6. Operacje na systemach plików 124](#_Toc1395864370)

[6.1. Tworzenie plików i katalogów 125](#_Toc177377362)

[6.1.1. Zadania 125](#_Toc461539594)

[6.2. Przeszukiwanie zawartości katalogu 126](#_Toc1618430515)

[6.2.1. Zadania 126](#_Toc1983803798)

[6.3. Praca z plikami tekstowymi 127](#_Toc1903111779)

[6.3.1. Zadania 127](#_Toc1126146782)

[6.4. Operacje na danych czasowych 128](#_Toc362438025)

[6.4.1. Zadania 128](#_Toc244736983)

[6.5. Podsumowanie 129](#_Toc1189799835)

[6.6. Zadania 129](#_Toc1015870251)

[6.7. Dodatek 131](#_Toc1670637774)

# Podstawy języka Python

E-skrypt stanowi kompleksowy materiał dydaktyczny, obejmujący 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 programów. Zadania o zróżnicowanym poziomie trudności, w tym bardziej wymagające, umożliwiają utrwalenie wiedzy i rozwój umiejętności w rozwiązywaniu problemów programistycznych, przygotowując do pracy z językiem Python na poziomie praktycznym i teoretycznym.

## 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 oraz podsumowanie, które pomaga utrwalić zdobytą wiedzę.

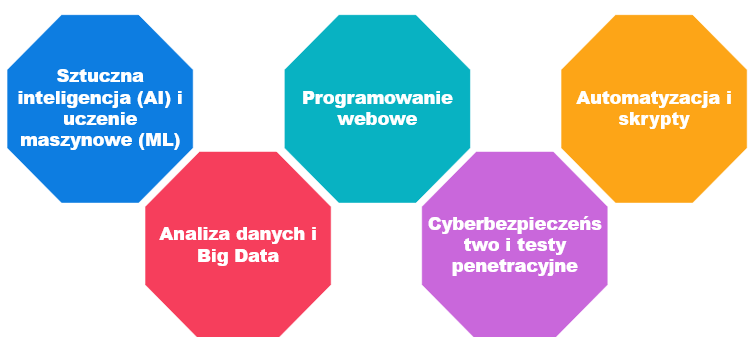
Sposób nauki opiera się na realizacji przykładów oraz samodzielnym wykonywaniu zadań utrwalających, co pozwala na aktywne przyswajanie materiału. Dla osób bardziej zaawansowanych przygotowano dodatkowe zadania (oznaczone \*) o wyższym stopniu trudności, które umożliwiają pogłębienie wiedzy i rozwój umiejętności 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 (*Rysunek 1.1)*. 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.



Rysunek 1.1. Zastosowania 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 (*Tabela 1.1*, *Tabela 1.2*), 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.

Tabela 1.1. 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.

Tabela 1.2. 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?

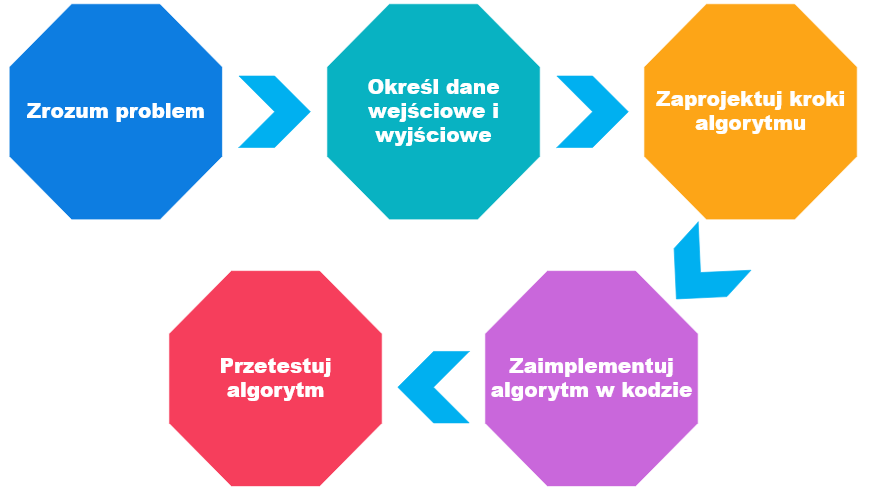
Algorytm to uporządkowany zbiór kroków lub instrukcji prowadzących do rozwiązania określonego problemu. Może być zapisany w postaci schematu blokowego, opisu słownego lub kodu komputerowego i powinien być jednoznaczny oraz efektywny. W języku Python algorytmy można implementować przy użyciu struktur 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.



Rysunek 1.2. 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

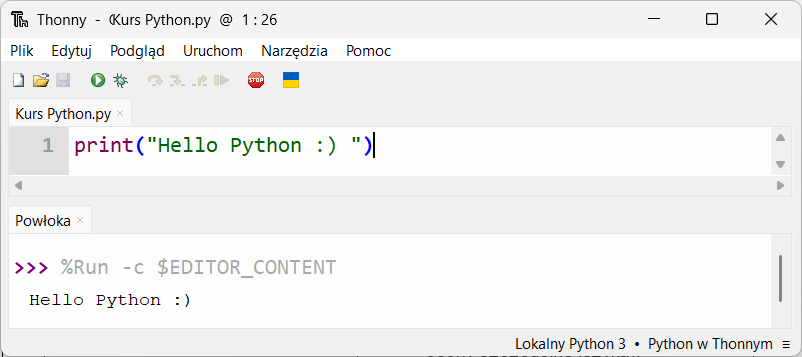
W trakcie kursu uczestnicy będą korzystać z różnych narzędzi wspomagających naukę i pracę z językiem Python. Obejmują one zarówno systemy operacyjne, na których działa Python, jak i środowiska programistyczne oraz edytory, które ułatwiają pisanie i uruchamianie kodu. W szczególności uczestnicy zapoznają się z pracą w konsoli systemowej (terminal w Linux, CMD/PowerShell w Windows), co pozwoli im zrozumieć działanie interpretera języka Python oraz 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.

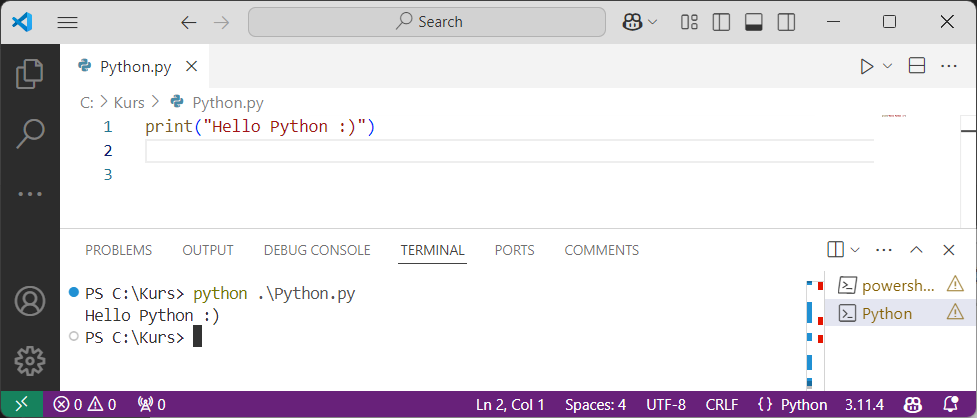
#### Środowiska i edytory (Thonny, Visual Studio Code)

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 (Rysunek 1.3). 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.



Rysunek 1.3. 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 (Rysunek 1.4). VS Code jest rozwijany przez firmę Microsoft i udostępniany na licencji open-source MIT, co pozwala na jego darmowe używanie oraz modyfikowanie przez społeczność.

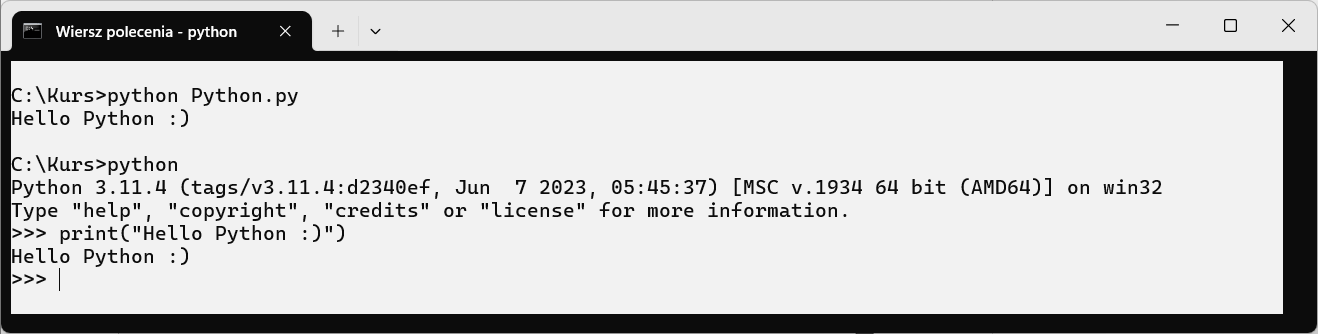


Rysunek 1.4. Visual Studio Code (VS Code) – lekki edytor z obsługą rozszerzeń

Do popularnych narzędzi powszechnie stosowanych do tworzenia aplikacji w języku Python należą 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.



Rysunek 1.5. 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 Linux i sprawdź uprawnienia (polecenie *whoami* pokaże Twoje UID, GID oraz grupy, do których należysz. Jeśli UID wynosi 0 (np. *uid=0(root))*, oznacza to, że działasz jako użytkownik root).

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

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

## Instalacja środowiska Python 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.

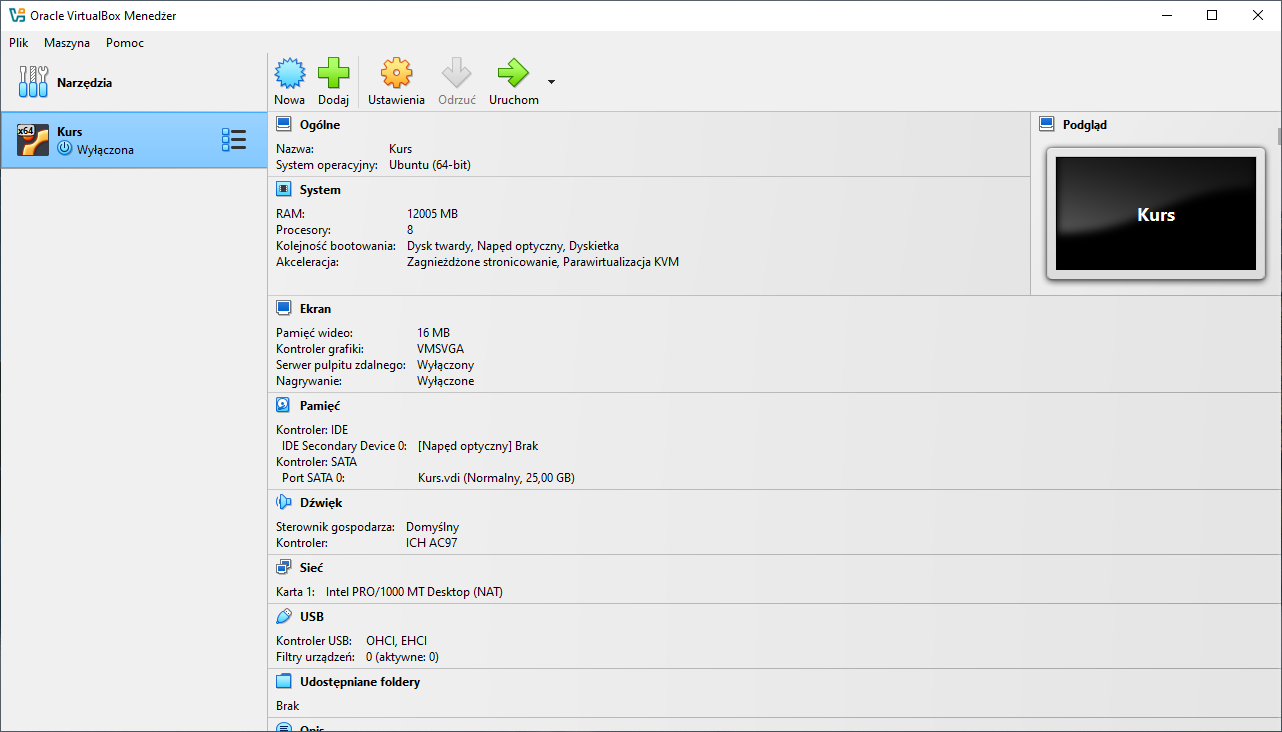
Aby skorzystać z systemu, należy uruchomić aplikację z poziomu pulpitu lub wpisać jej nazwę w pasku wyszukiwania systemu (Rysunek 1.6, Rysunek 1.1).

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

Rysunek 1.6. Ikona emulatora systemu Oracle VirtualBox

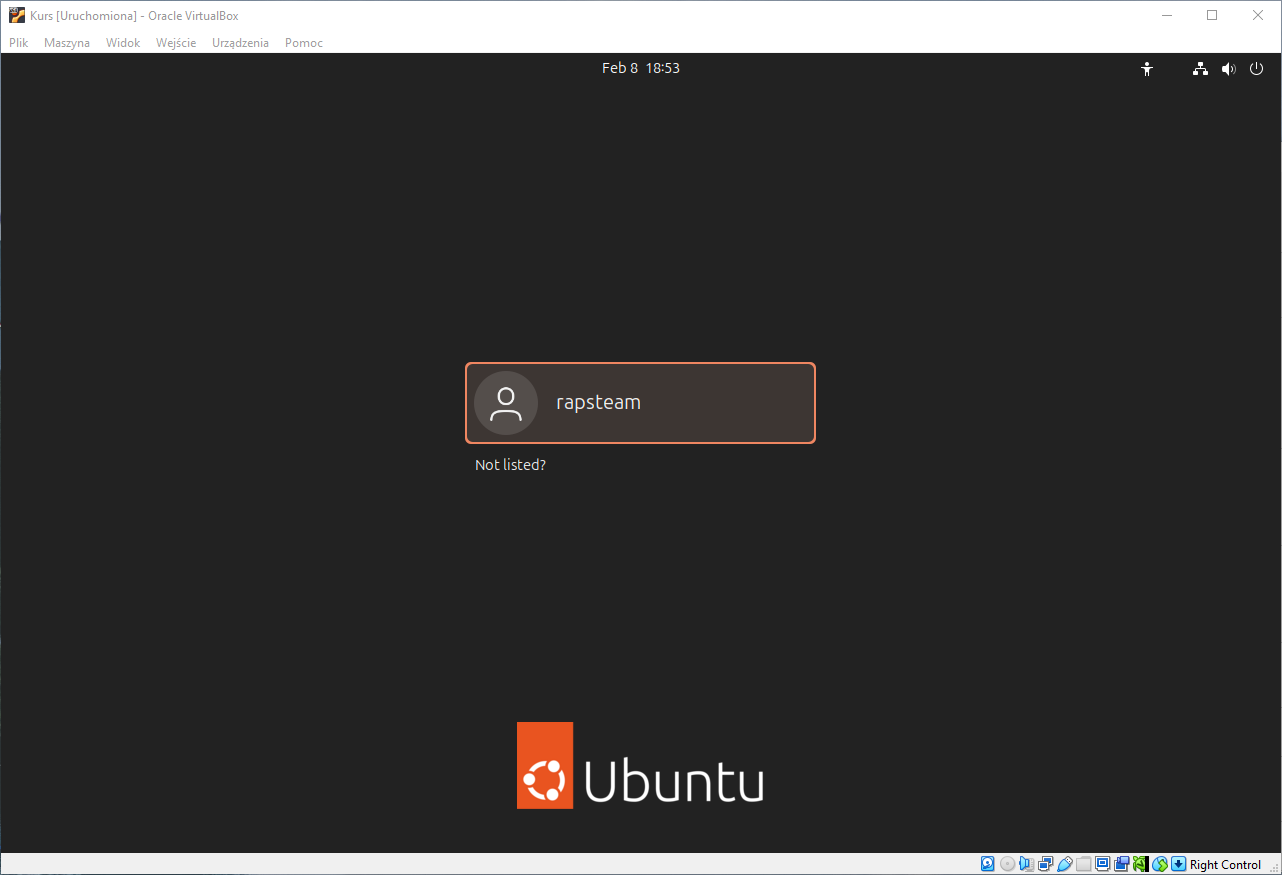
Po uruchomieniu, pojawi się okno z widokiem dostępnych i skonfigurowanych maszyn wirtualnych, gotowych do uruchomienia. Aby wystartować jedną z nich, należy wybrać pozycję z listy po prawej stronie widoku o nazwie *Kurs* i nacisnąć klawisz *Uruchom* widoczny w górnej części okna.

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.



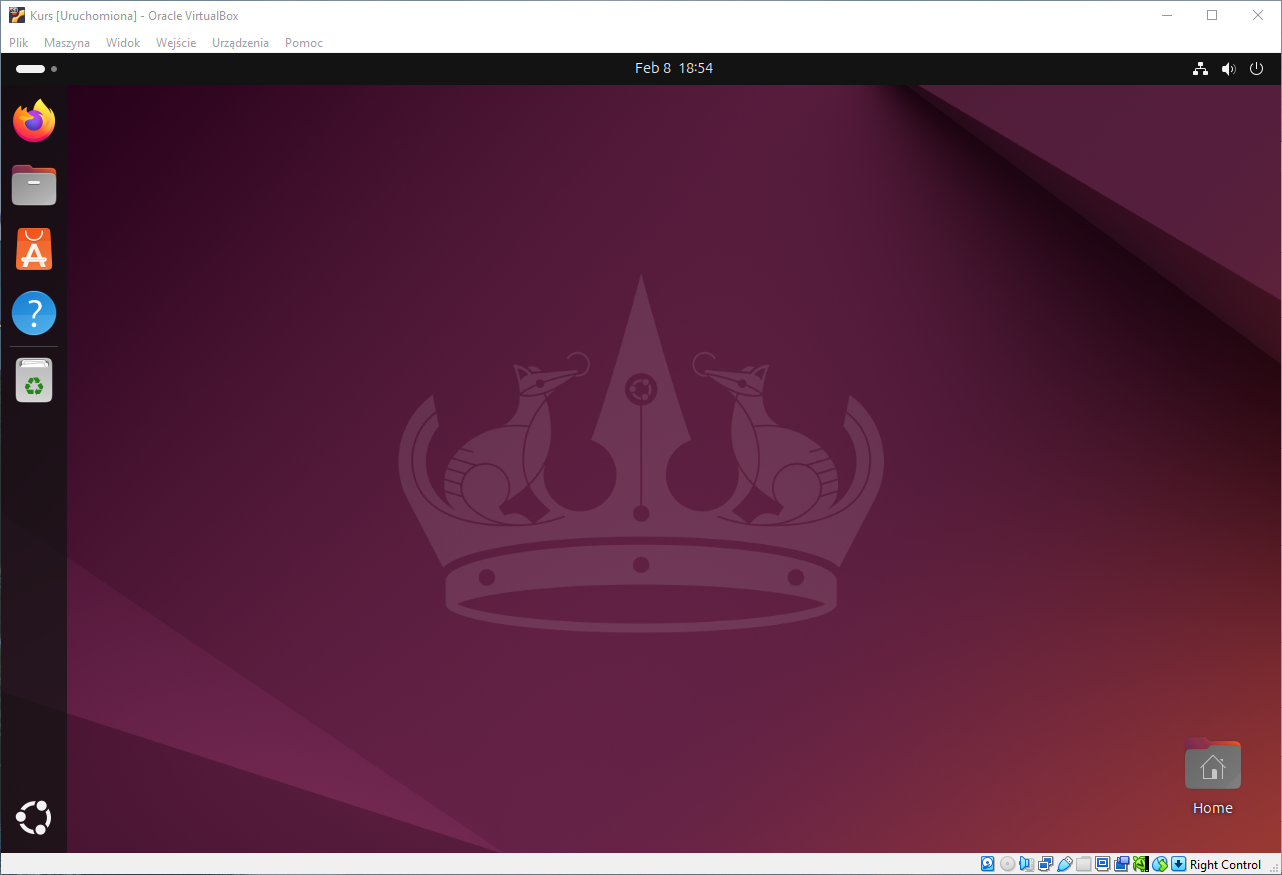
Rysunek 1.7 Okno wyboru skonfigurowanych maszyn wirtualnych

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.



Rysunek 1.8 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.

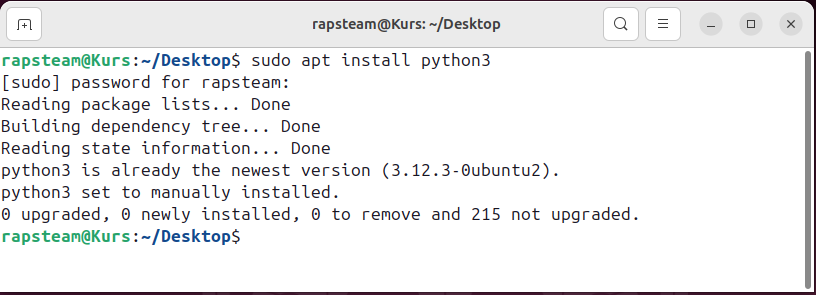


Rysunek 1.9 Pulpit systemu Ubuntu

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.

Komenda 1.1 Instalacja interpretera Python 3 w systemie Ubuntu

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



Rysunek 1.10 Efekt instalacji pakietu Python w systemie Ubuntu

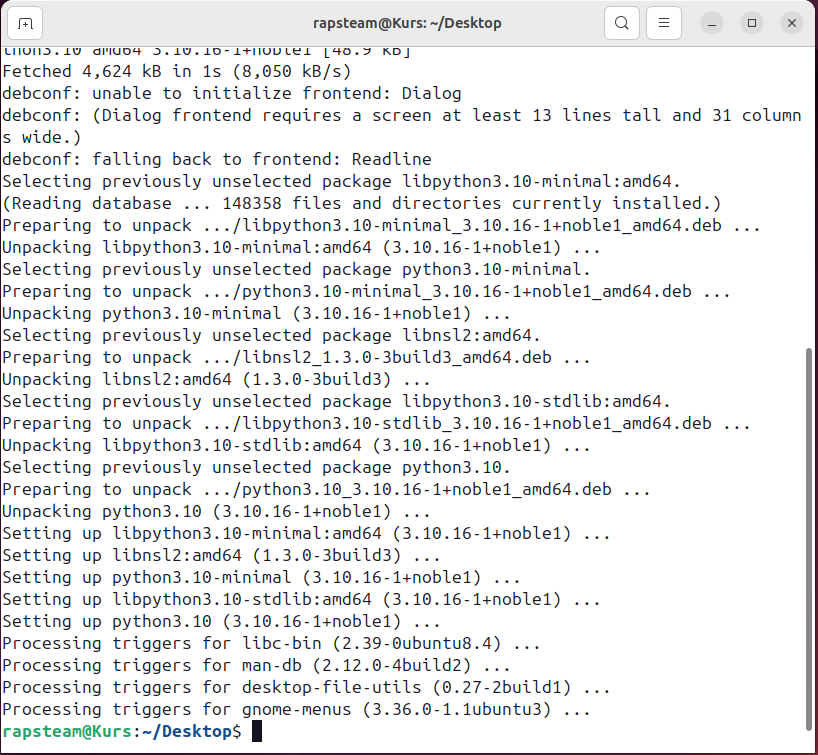
Możliwe jest także wskazanie konkretnej wersji pakietu Python do instalacji. Dzięki instalacji wielu wersji pakietu, można zapewnić kompatybilność różnych projektów, które wymagają specyficznych wersji interpretera i bibliotek. Pozwala to uniknąć konfliktów między zależnościami oraz zapewnia stabilność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

Komenda 1.2 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*. |

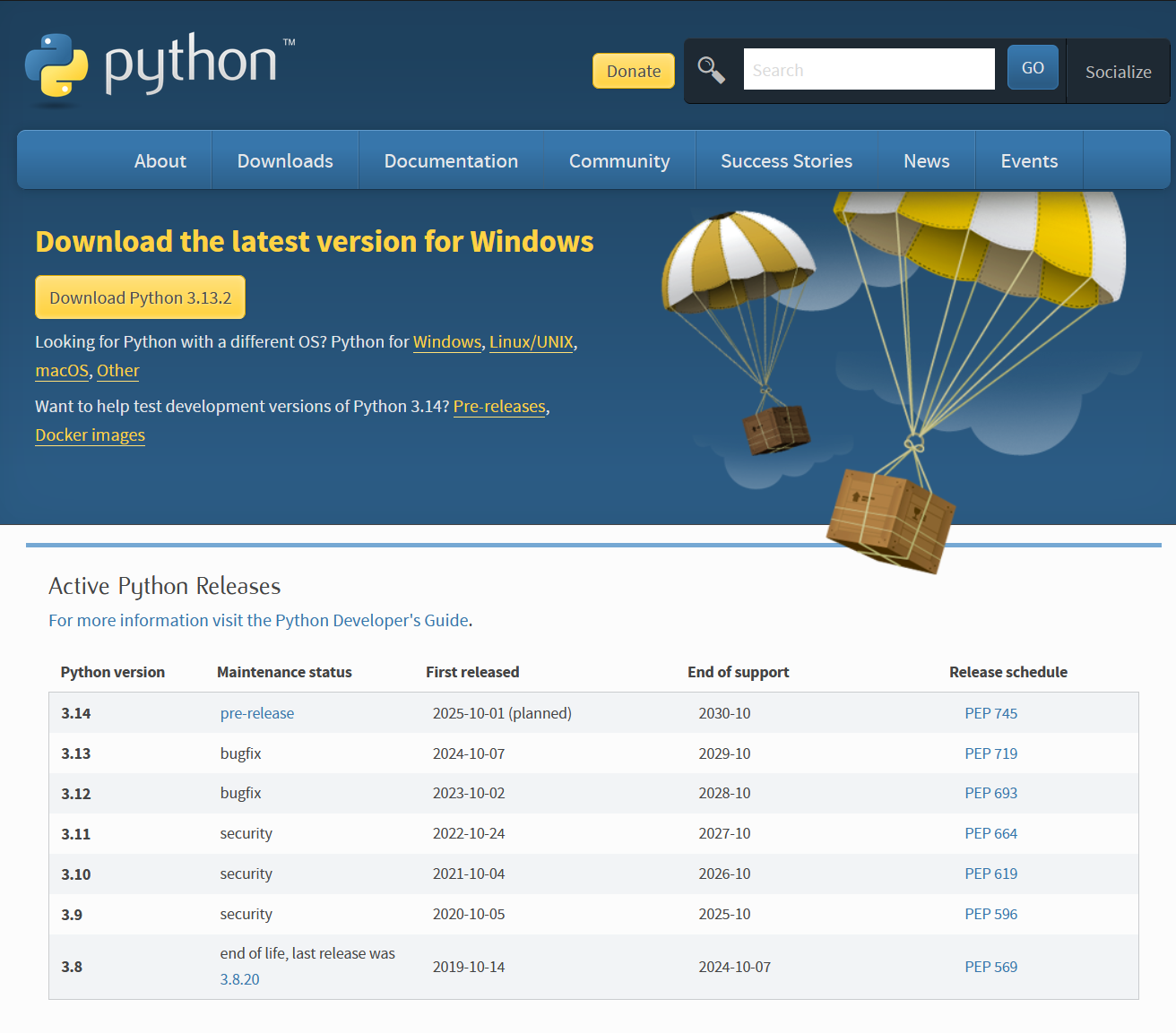


Rysunek 1.11. Proces instalacji pakietu Python w systemie Ubuntu

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.



Rysunek 1.12. Oficjalna strona projektu Python

|  |  |
| --- | --- |
|  | Na ekranie Rysunek 1.12 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 (Rysunek 1.13) Do dyspozycji są dwie opcje instalacji:

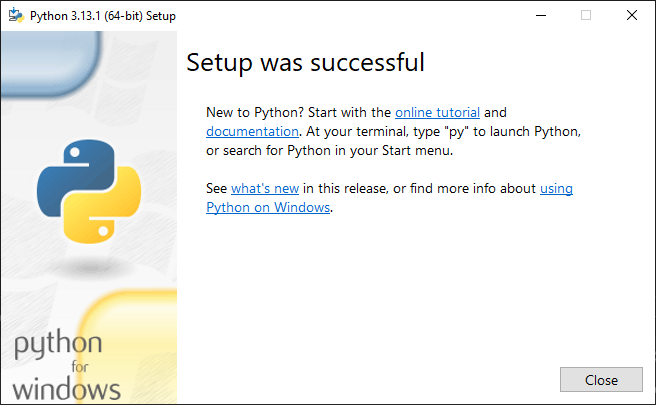
* Instalacja uproszczona (*Install Now*) – instaluje interpreter w domyślnej, wskazanej przez program lokalizacji systemowej oraz załącza do tego procesu szereg podstawowych, najpopularniejszych bibliotek dla tego języka, menadżer pakietów PIP oraz ustawia skojarzenie plików z rozszerzeniem *.py* jako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



Rysunek 1.13. 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.



Rysunek 1.14. Ekran końcowy instalatora Python potwierdzający zakończenie instalacji pomyślnym komunikatem.

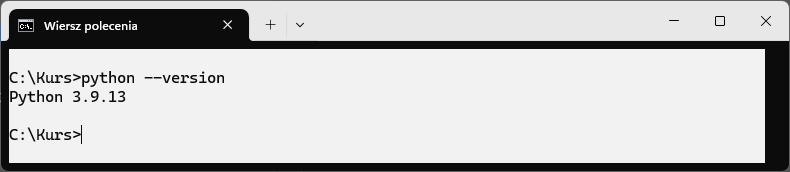
### Weryfikacja wersji interpretera języka Python

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. |

Komenda 1.3 Wyświetlenie obecnie zainstalowanej wersji interpretera Python

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



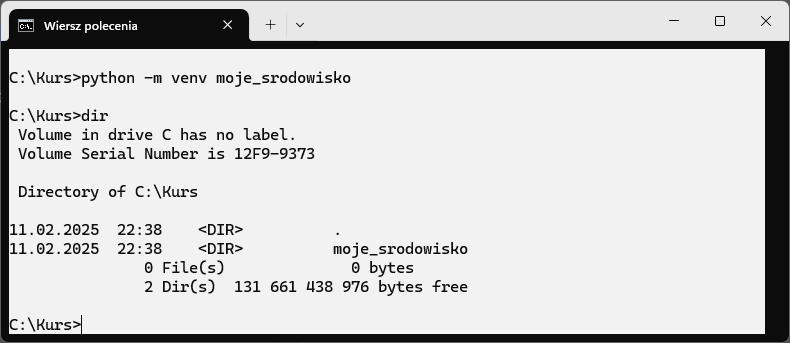
Rysunek 1.15. Weryfikacja instalacji Python w systemie Windows

### 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* (Rysunek 1.16), aby zainicjować proces jego tworzenia.

Komenda 1.1. Komenda wywołująca stworzenie środowiska wirtualnego dla języka Python

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

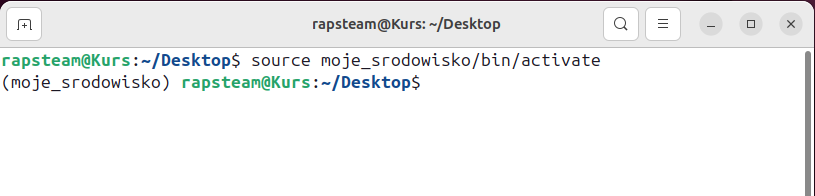


Rysunek 1.16. Tworzenie wirtualnego środowiska za pomocą pakietu venv

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

Listing 1.4. Polecenie aktywujące środowisko wirtualne w systemie Ubuntu

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

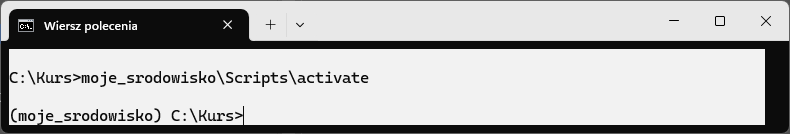


Rysunek 1.17. Wyłączenie środowiska wirtualnego w systemie Ubuntu

Po utworzeniu środowiska należy je aktywować. W dystrybucji Debian aktywację wykonuje się poleceniem *source* (Listing 1.4)*,*  natomiast w systemie Windows uruchamiając skrypt *activate* (Listing 1.5).

Listing 1.5. Polecenie aktywujące środowisko wirtualne w systemie Windows

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



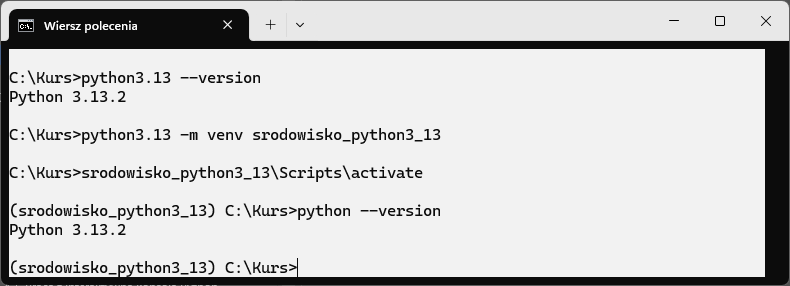
Rysunek 1.18. Efekt aktywacji środowiska wirtualnego (win)

Zmiana wyglądu tzw. promptu (Rysunek 1.8) 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.



Rysunek 1.19 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.

### 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 Aktywacja i dezaktywacja środowiska wirtualnego**

* Aktywuj środowisko z poprzedniego zadania a następnie je deaktywuj.

**Zadanie 1.2.4 Specyficzne wersje Python**

* Stwórz środowisko wirtualne języka Python dla specyficznej wersji języka.

**Zadanie 1.2.5\* Izolowane środowiska Python**

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

## Zasady tworzenia i struktury skryptów w języku Python

Pliki z rozszerzeniem *.py* są podstawowymi jednostkami kodu w języku Python. Są to pliki tekstowe zawierajce instrukcje języka Python, takie jak definicje funkcji, klasy, zmienne oraz importy modułów. Można je edytować w dowolnym edytorze tekstowym. Znaczenie poszczególnych elementów, które mogą znaleźć się w pliku, zostanie omówione w kolejnych rozdziałach. W tym rozdziale skupimy się wyłącznie na poprawnym zapisie kodu w plikach *.py.*

### 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.

Listing 1.1. Przykładowy plik języka Python .py z poprawnymi wcięciami

|  |
| --- |
| [module\_1\_lesson\_2\_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 zodstanie omówiona w dalszych rozdziałach. |

Listing 1.2. Skrypt języka Python załamaniami lini

|  |
| --- |
| [module\_1\_lesson\_2\_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, Notatniku wybierz nowy plik tekstowy i uzupełnij go treścią kodu zawartego w Listing 1.3. Następnie zapisz plik na Pulpicie z rozszerzeniem *.py*.

Listing 1.3. Program wsypujący „Witaj w świecie Pythona!”

|  |
| --- |
| [module\_1\_lesson\_2\_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.

Listing 1.4. 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.

Listing 1.5. Kod umożliwiający przekazywanie argumentów w wierszu poleceń

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

### Wybór trybu pracy: interaktywny czy skryptowy?

W rozdziale 1 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.



Rysunek 1.20. 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.



Rysunek 1.21. 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.

Listing 1.6. Prosty kod, który można uruchamiać wielokrotnie

|  |
| --- |
| [module\_1\_lesson\_2\_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.

Listing 1.7. Kod automatyzujący zapis aktualnej daty i godziny

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

Tabela 1.3. 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.

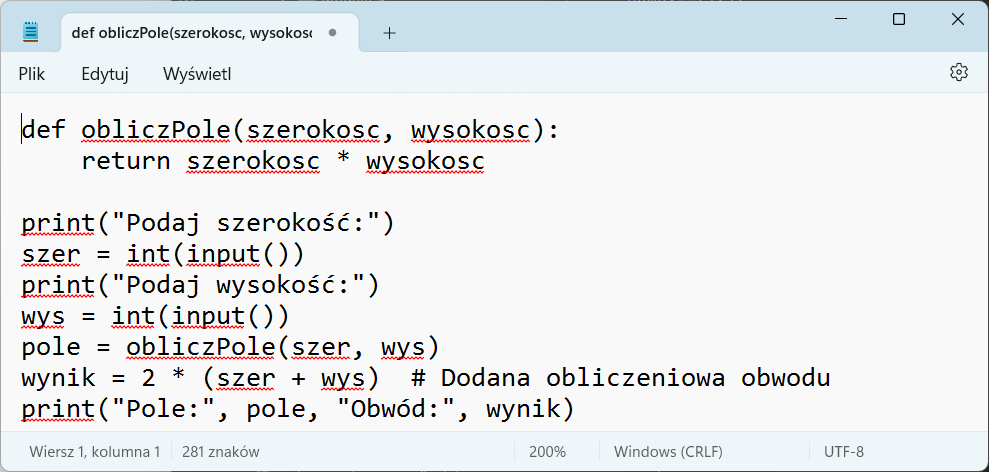
### Zadania

**Zadanie 1.3.1. 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 1.3.2. Popraw formatowanie kodu**

* Przepisz kod stosując poprawne formatowanie kodu oraz zapisz skrypt.



**Zadanie 1.3.3. Obsługa argumentów wiersza poleceń**

* Napisz skrypt, który przyjmuje argument podany w wierszu poleceń i wypisuje go na ekranie.
* Jeśli użytkownik nie poda argumentu, program powinien wyświetlić komunikat "Nie podano argumentu!".

**Zadanie 1.3.4. \* 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").

**Zadanie 1.3.5. \* Automatyczne zapisywanie daty do pliku**

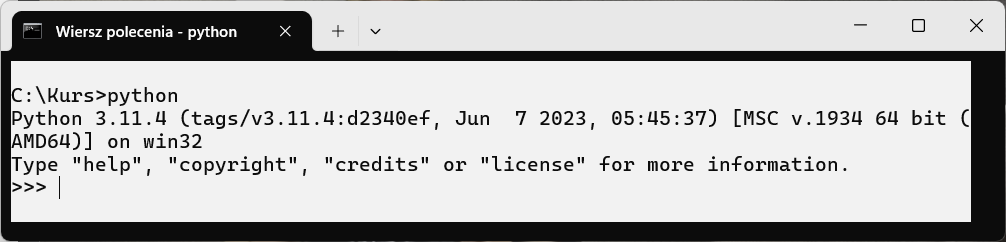
* Napisz skrypt, który pobiera aktualną datę i godzinę i zapisuje ją do pliku *log.txt*.
* Przykładowy format zapisu *2025-02-05 14:30:12 - Uruchomiono skrypt*
* Jeśli plik *log.txt* nie istnieje, skrypt powinien go utworzyć.

## Uruchamianie kodu w konsoli za pomocą interpretera

W tym rozdziale skupimy się na trybie interaktywnym języka Python, który jest niezwykle przydatny do testowania fragmentów kodu, eksplorowania bibliotek oraz nauki języka. Umożliwia on wprowadzanie pojedynczych komend i natychmiastowe obserwowanie wyników, co pozwala szybko sprawdzać działanie funkcji, operacji matematycznych czy struktur danych bez konieczności tworzenia pełnych skryptów.

### 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.



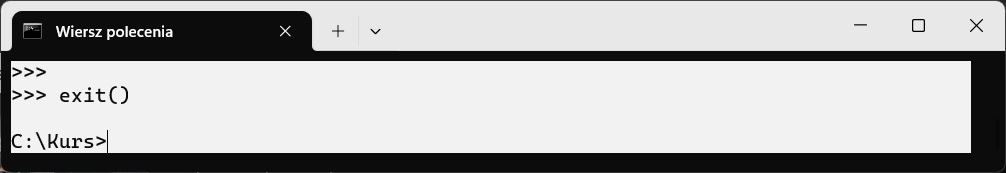
Rysunek 1.22. Uruchomienie trybu interaktywnego dla domyślnej wersji Python

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*.

Listing 1.8. 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.

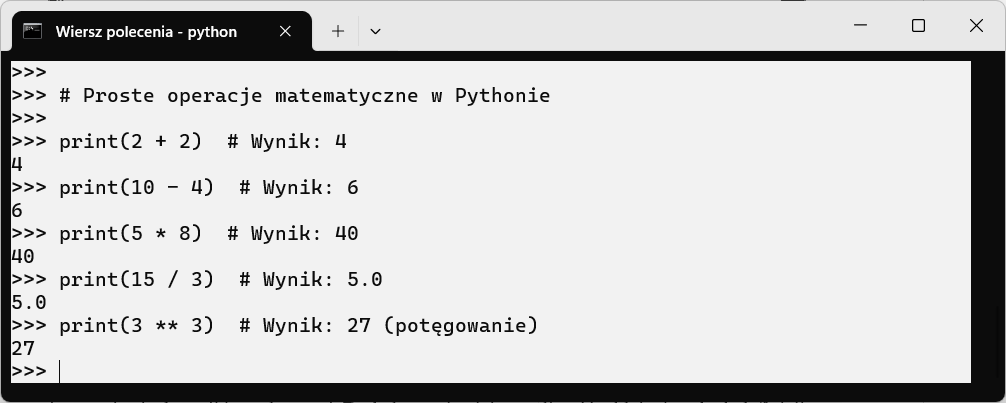


Rysunek 1.23. Wyłączenie trybu interaktywnego

### 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.



Rysunek 1.24. Natychmiastowe obserwowanie wyników w trybie interaktywnym

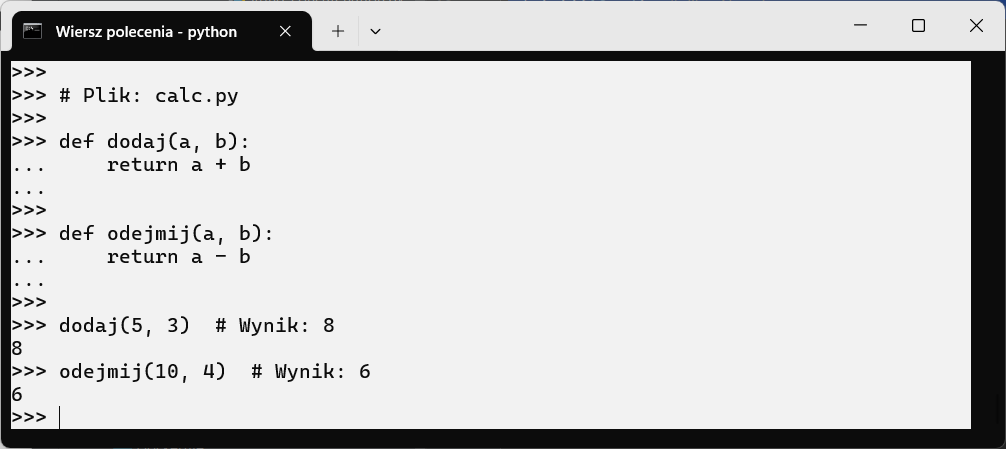
### Testowanie fragmentów kodu w trybie interaktywnym

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.

Listing 1.9. Wielolinijkowy fragment kodu

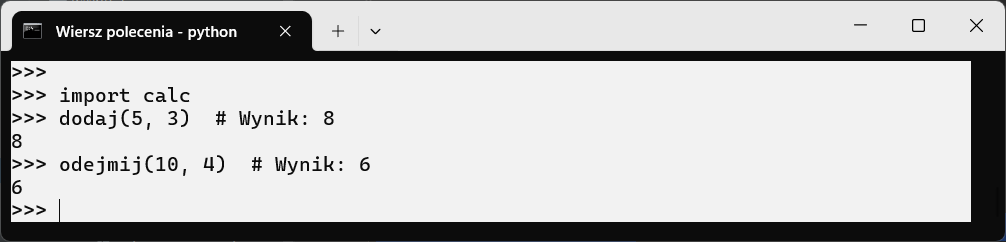
|  |
| --- |
| [module\_1\_lesson\_3\_calc.py] |

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.



Rysunek 1.25. 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.



Rysunek 1.26. Testowanie fragmentów kodu z pliku .py w konsoli

### Zadania

**Zadanie 1.4.1. 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 1.4.2. 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 1.4.3. Importowanie i testowanie modułów**

* Zaimportuj wbudowaną bibliotekę: *import math wypisz zawartosć* *math.pi.*

**Zadanie 1.4.4. \* 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 1.4.5. \* Dynamiczne ładowanie modułu w Pythonie**

*Cel: Celem zadania jest zrozumienie, jak importować i ponownie ładować zmodyfikowane moduły w trybie interaktywnej konsoli bez konieczności restartowania interpretera.*

* Napisz skrypt *dynamic\_module.py*, który zawiera funkcję *powitaj(imie),* zwracającą powitanie z podanym imieniem.
* Uruchom tryb interaktywny, zaimportuj moduł i przetestuj działanie funkcji.
* Dodaj nową funkcję *policz\_litery(tekst),* spróbuj ją wywołać.
* Następnie wymuś ponowne załadowanie modułu za pomocą *importlib*, a następnie ponownie przetestuj działanie funkcji.

|  |  |
| --- | --- |
|  | Aby ponownie załadować moduł, należy zaimportować bibliotekę *importlib*, a następnie wykonać komendę *importlib.reload(nazwa)*. |

## Prowadzenie operacji matematycznych i manipulacji tekstowych

W tej lekcji zajmiemy się podstawowymi operacjami matematycznymi i manipulacją napisami w języku Python. Nauczymy się wykonywać działania arytmetyczne, takie jak dodawanie, mnożenie czy potęgowanie, a także pracować z tekstem poprzez łączenie, powielanie i zmianę wielkości liter. Dodatkowo poznamy wbudowane funkcje *len()* i *type()*, które pozwalają sprawdzać długość napisów oraz typy danych.

### 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.

Tabela 1.4. 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 Listing 1.10, gdzie zaprezentowano ich praktyczne zastosowanie w języku Python.

Listing 1.10. Przykłady podstawowych operacji arytmetycznych

|  |
| --- |
| [module\_1\_lesson\_3\_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.

Listing 1.11. Przykład użycia operatorów skróconych dodawania oraz potęgowania

|  |
| --- |
| [module\_1\_lesson\_3\_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.

Listing 1.12. Operacje matematyczne na liczbach całkowitych i zmiennoprzecinkowych

|  |
| --- |
| [module\_1\_lesson\_3\_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()*.

Tabela 1.5. 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" |

Listing 1.13. Przykłady użycia metod łączenia napisów

|  |
| --- |
| [module\_1\_lesson\_3\_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.

Tabela 1.6. 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" |

Listing 1.14. Przykład użycia powielania napisów

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

**Zmiana wielkości liter**

Python oferuje wbudowane metody do zmiany wielkości liter w napisach. Tabela 1.1 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.

Tabela 1.7. 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!" |

Listing 1.15. Przykład użycia metod zmiany wielkości liter

|  |
| --- |
| [module\_1\_lesson\_3\_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.

Tabela 1.8. 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.

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

### Zadania

**Zadanie 1.5.1. Suma oraz różnica**

* Napisz program, który wykona obliczenie 5+2 oraz 5-2.

**Zadanie 1.5.2. Mnożenie i dzielenie**

Napisz program, który wykona obliczenie 5\*2 oraz 5/2 i wypisze typy wyników.

**Zadanie 1.5.3. Potęgowanie**

Napisz program, który wykona podniesie liczbę 2 do potęgi 8.

**Zadanie 1.5.4. Łączenie napisów**

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

**Zadanie 1.5.5. \* 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 zmiennych języka Python

Zmienna to nazwana etykieta odnosząca się do obszaru pamięci, w którym przechowywane są dane. Jej wartość może się zmieniać, co umożliwia dynamiczne operacje w programie i elastyczne przetwarzanie informacji. W programowaniu zmienne są podstawowym narzędziem do przechowywania wyników obliczeń i danych wejściowych. W języku Python przypisanie wartości do zmiennej odbywa się za pomocą operatora *=*, np. *a = 5*.

Listing 1.14 Przypisanie wartości 5 do zmiennej o nazwie a

|  |
| --- |
| >>> a = 5 |

W języku Python zmienne nie posiadają jawnej deklaracji typu. W momencie przypisania wartości do zmiennej, interpreter automatycznie ustala, jaki typ danych reprezentuje ta wartość. Proces ten odbywa się poprzez proste przypisanie, na przykład *x = 10*, co powoduje, że zmienna *x* przyjmuje typ całkowity, lub *tekst = "przykład"*, gdzie *teks*t staje się zmienną typu ciąg znaków.

System typów w języku Python jest dynamiczny oraz silnie typowany, co oznacza, że choć zmienne mogą zmieniać przypisane im wartości, operacje na nich muszą być zgodne z określonymi regułami dotyczącymi typów. Język posiada zestaw typów danych, które można podzielić na kilka głównych kategorii.

Tabela: Hierarchia typów standardowych w Pythonie

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hierarchia typów standardowych w Pythonie | | | | |
| **Liczbowe** | **Sekwencyjn**e | **Zbiory** | **Mapując**e | **Braku wartości** |
| Całkowite (int)  Zmiennoprzecinkowe (float)  Zespolone (complex)  Logiczne (bool) | Łańcuchy znaków (str)  Listy (list)  Krotki (tuple)  Sekwencje bajtów (bytes/bytearrays) | Zbiory (set) | Słowniki (dict) | None |

### 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. Za pomocą funkcji *int()* można konwertować różne typy danych, takie jak *float*, *str* i *bool*.



Rysunek 1.27 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. W praktyce, *float* pozwala na wykonywanie obliczeń wymagających reprezentacji wartości dziesiętnych, jednak należy pamiętać o typowych ograniczeniach wynikających z reprezentacji binarnej. Analogicznie jak w przypadku *int*, za pomocą funkcji *float()* można dokonywać konwersji typów *int, str* i *bool* na typ zmiennoprzecinkowy *float.*



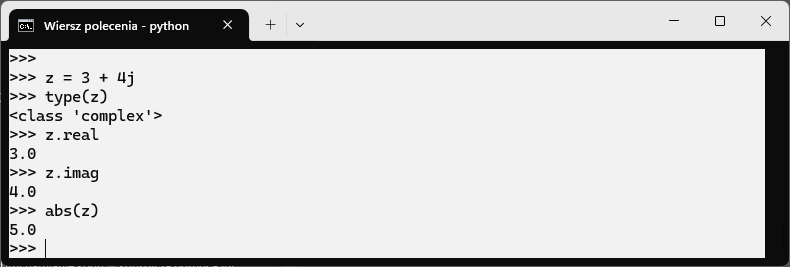
Rysunek 1.28 Przykłady liczb zmiennoprzecinkowych



Rysunek 1.29 Przykład błędu zaokrąglenia liczb zmiennoprzecinkowych

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

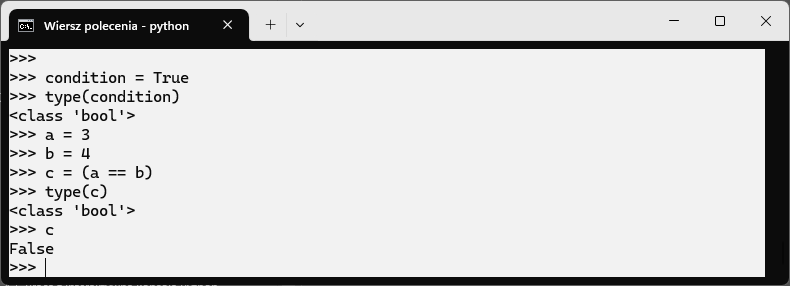
Składają się z części rzeczywistej oraz urojonej. Notacja liczby zespolonej w Pythonie wykorzystuje literę j do oznaczenia części urojonej, na przykład zapis *3 + 4j* oznacza liczbę zespoloną, w której część rzeczywista wynosi 3, a część urojona 4. Typ complex umożliwia wykonywanie operacji arytmetycznych specyficznych dla liczb zespolonych, co jest przydatne w zastosowaniach matematycznych i inżynieryjnych. Za pomocą funkcji *complex()* można dokonywać konwersji typów *int, float* i *str* na typ liczby zespolonej *complex*.



Rysunek 1.30 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.



Rysunek 1.31 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. Aby utworzyć *str*, wystarczy przypisać do zmiennej ciąg znaków ujęty w pojedyncze, podwójne lub potrójne cudzysłowy. Użycie pojedynczych lub podwójnych cudzysłowów umożliwia definiowanie krótkich tekstów, natomiast potrójne cudzysłowy pozwalają na tworzenie ciągów wieloliniowych, zachowując formatowanie oryginalnego tekstu.

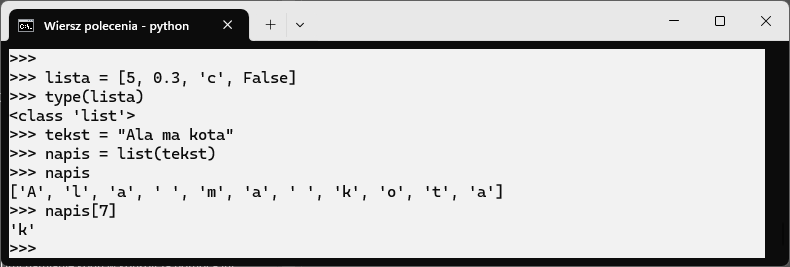


Rysunek 1.32 Przykład stworzenia ciągu znaków w postaci zdania, oraz pojedynczego znaku

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

**Listy (list)**

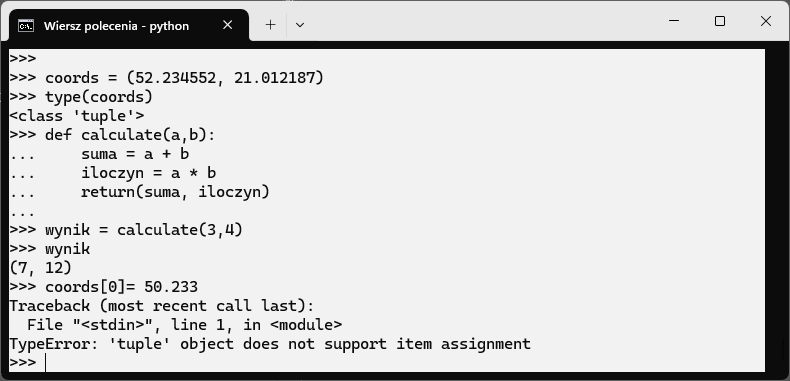
Stanowią mutowalną sekwencję, co oznacza, że zawartość listy może być modyfikowana po jej utworzeniu. W liście można przechowywać elementy różnych typów, a ich kolejność jest zachowywana. Operacje takie jak dodawanie, usuwanie czy modyfikacja elementów są możliwe i wpływają na stan listy. Notacja listy polega na umieszczeniu elementów w nawiasach kwadratowych, oddzielonych przecinkami. Numerowanie elementów w liście rozpoczyna się od elementu zerowego.



Rysunek 1.33 Przykład utworzenia listy zawierającej wiele typów danych

**Krotki (tuple)**

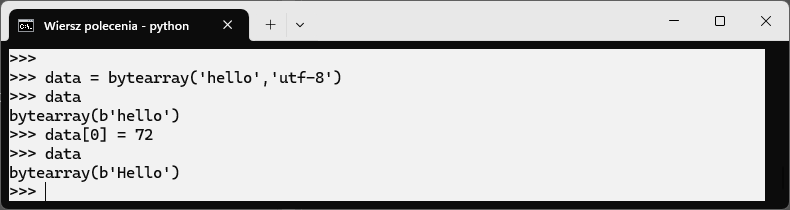
Krotki to sekwencje, podobne do list, ale raz utworzone nie można ich zmienić. Oznacza to, że dane zapisane w krotce pozostają stałe, co jest przydatne, gdy potrzebujemy pewności, że zestaw informacji nie zostanie przypadkowo zmodyfikowany. Aby utworzyć krotkę, wystarczy umieścić elementy w nawiasach okrągłych lub oddzielić je przecinkami (nawiasy są opcjonalne). Dzięki temu, że krotki są niezmienne, można je wykorzystywać jako klucze w słownikach lub jako elementy zbiorów, pod warunkiem, że same zawierają tylko niezmienne dane. Dodatkowo krotki są często używane do zwracania więcej niż jednego wyniku z funkcji.



Rysunek 1.34 Niezmienność krotek w języku Python

**Sekwencja bajtów (bytes/bytearray)**

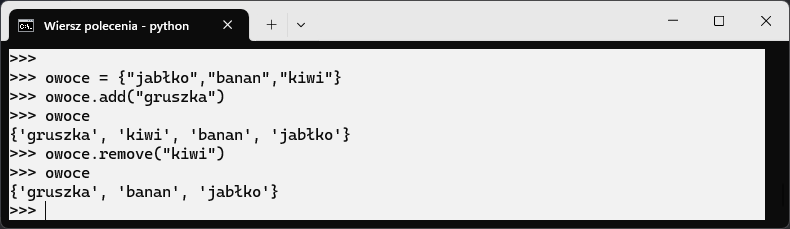
Sekwencyjne typy reprezentujące dane binarne. Typ *bytes* jest niemutowalnym ciągiem liczb całkowitych, gdzie każda liczba mieści się w przedziale od 0 do 255, i jest wykorzystywany do reprezentowania danych binarnych. Wersja mutowalna, *bytearray*, umożliwia modyfikację zawartości po jej utworzeniu,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



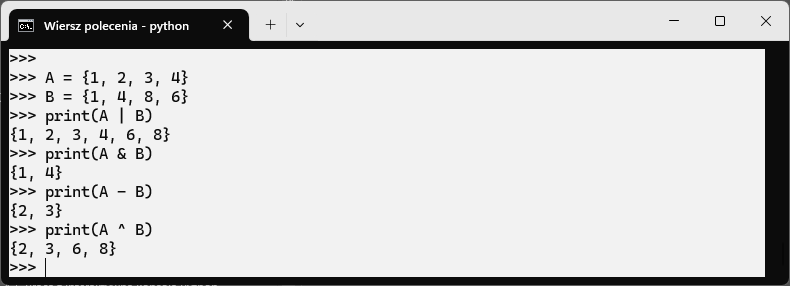
Rysunek 1.35 Przykład utworzenia krotki oraz stowrzenia jej jako zwracany typ funkcji

### Zbiory

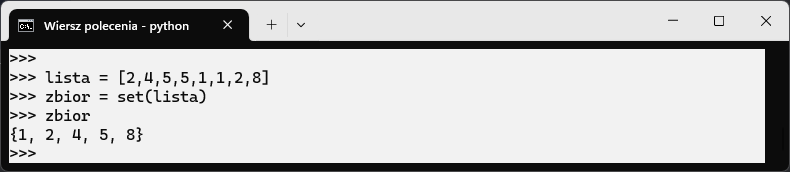
Zbiór tworzony jest poprzez ujęcie elementów w nawiasy klamrowe, na przykład: *{1, 2, 3},* lub za pomocą funkcji *set(),* która konwertuje iterowalne obiekty na zbiór. Zbiór umożliwia wykonywanie operacji matematycznych, takich jak suma, różnica, przecięcie czy symetryczna różnica, co przydaje się przy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



Rysunek 1.36. Operacja utworzenia zbioru i dodania i usunięcia jego elementu



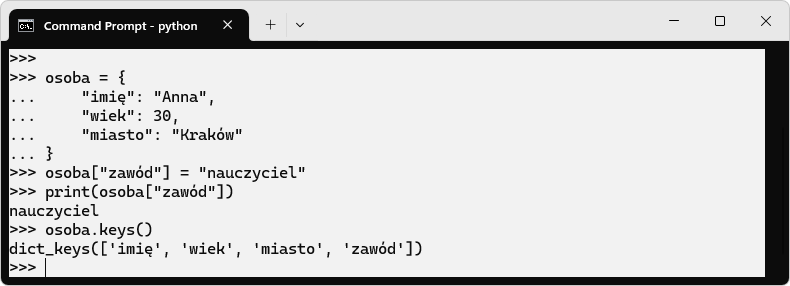
Rysunek 1.37. Podstawowe operacje na zbiorach



Rysunek 1.38 Usuwanie duplikatów i sortowanie elementów listy poprzez rzutowanie

### Typ mapujący

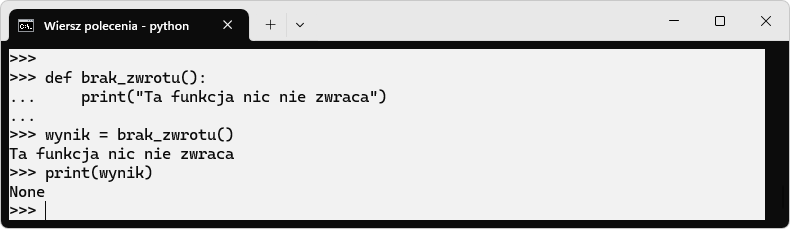
Słownik (dictionary) to jeden z typów mapujących języka Python, oparty na unikalnych kluczach niemutowanego typu, które są przypisane do odpowiadających im wartości. Elementy słownika zapisuje się w nawiasach klamrowych w formie par klucz-wartość, oddzielonych dwukropkiem. Słowniki umożliwiają szybki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



Rysunek 1.39 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.



Rysunek 1.40 Domyślny zwrot None w Pythonie

### Zadania

**Zadanie 1.6.1. Sprawdzenie typów utworzonych zmiennych**

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

**Zadanie 1.6.2. Zmienna None**

Utwórz zmieną x = None i wypisz jej typ.

**Zadanie 1.6.3. 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 1.6.4. Porównanie typów**

* Utwórz zmienne a = "10" oraz b = 5

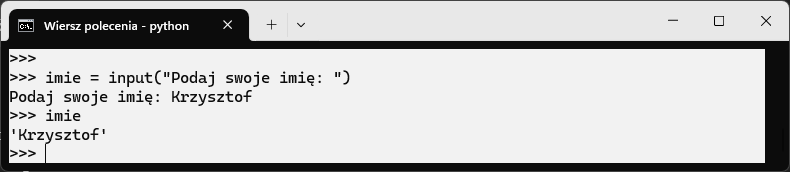
**Zadanie 1.6.5. \* Liczby zespolone**

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

## Używanie metod wejścia i wyjścia

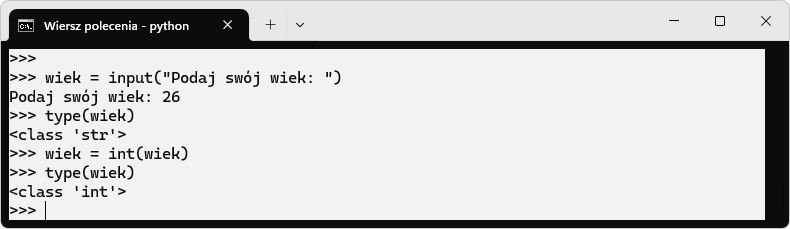
### Funkcja input()

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.*



Rysunek 1.41 Pobranie od użytkownika danych za pomocą funkcji input()

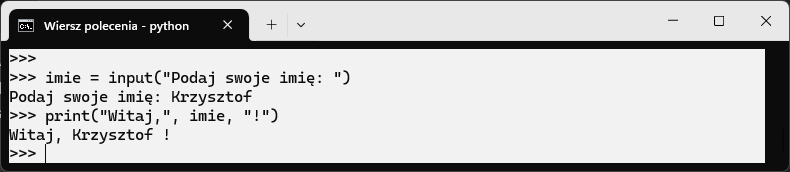
Warto zauważyć, że funkcja *input()* zawsze zwraca wartość typu string, co oznacza, że przy pobieraniu liczb konieczna jest konwersja, na przykład przy użyciu funkcji *int()* lub *float()*:



Rysunek 1.42 Pobranie informacji od użytkownika i konwersja jej typu z str na int

### Funkcja print()

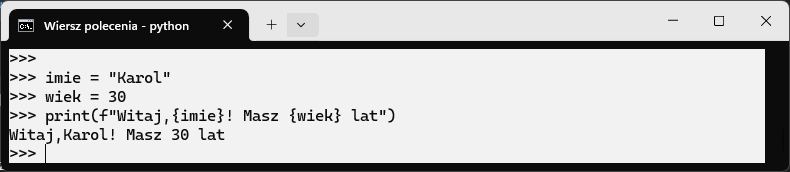
Funkcja *print()* służy do prezentowania wyników na ekranie poprzez standardowe wyjście. Argumenty przekazywane do funkcji są konwertowane na ciągi znaków, a następnie wyświetlane. Domyślnie kolejne argumenty oddzielane są spacją, a na końcu wyniku funkcja dodaje znak nowej linii. Przykładowe użycie 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.



Rysunek 1.43 Pobranie informacji od użytkownika i wyświetlenie jej za pomocą funkcji print()

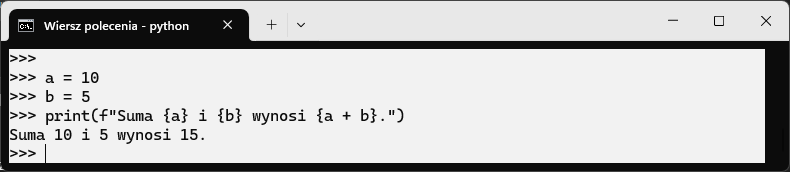
### Formatowanie stringów

Formatowanie tekstu w funkcji *print()* przy użyciu *f-stringów* wprowadza możliwość osadzania wyrażeń bezpośrednio w tekście. Tak zwane *f-stringi* dostępne są od wersji Python 3.6. Aby skorzystać z tej funkcjonalności, ciąg znaków poprzedza się literą *f* lub *F*, a w miejscach, gdzie mają być podstawione zmienne lub wyrażenia, używa się nawiasów klamrowych.



Rysunek 1.44 Przykład wyświetlenia danych typu str i int jako ciąg znaków wyjścia standardowego

W wyniku powyższego kodu zostanie wyświetlony sformatowany ciąg znaków, w którym wartości zmiennych *imie* i *wiek* zostaną podstawione do tekstu. *F-stringi* umożliwiają również wykonywanie prostych operacji wewnątrz nawiasów klamrowych, co pozwala na dynamiczne modyfikowanie prezentowanych danych.



Rysunek 1.45 Przykład operacji na zmiennych wewnątrz sformatowanego wyjścia.

W tym przypadku w nawiasach klamrowych obliczana jest suma zmiennych *a* i *b*, a wynik operacji jest wstawiany do ciągu tekstowego. Formatowanie tekstu przy użyciu *f-stringów* poprawia czytelność kodu oraz ułatwia tworzenie przejrzystych komunikatów wyjściowych, integrując tekst i dane w jednym ciągu znaków.

### Zadania

**Zadanie 1.7.1. Wczytaj imię i wiek**

* Napisz program, który poprosi użytkownika o jego imię i wiek, a następnie wyświetli komunikat w formacie:
  + *"Cześć [imię]! Masz [wiek] lat."*

**Zadanie 1.7.2. Ulubione kolory**

* Napisz program, który poprosi użytkownika o dwa ulubione kolory, a następnie wyświetli je w jednym zdaniu, używając f-stringa.

**Zadanie 1.7.3. 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 1.7.4. \* Obliczanie wieku użytkownika**

* Napisz program, który zapyta użytkownika o jego imię, nazwisko oraz rok urodzenia. Następnie wyświetli zdanie w formacie:
  + *"Nazywasz się [imię] [nazwisko] i masz [wiek] lat."*
* Uwaga: Program powinien obliczyć wiek na podstawie aktualnego roku.

**Zadanie 1.7.5. \* 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]."*

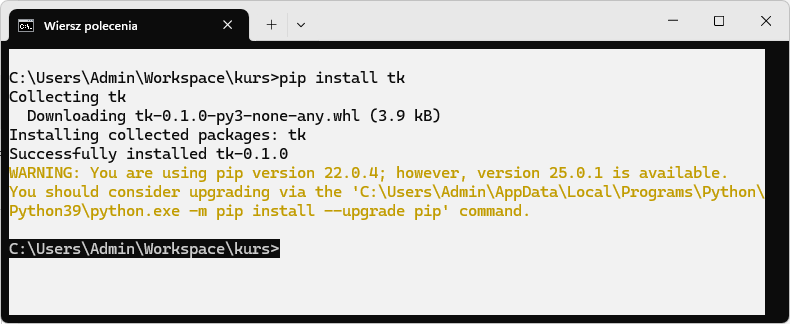
## Korzystanie z bibliotek i menadżera PIP

Biblioteki znacznie rozszerzają możliwości języka, przyspieszają pracę programisty oraz zapewniają bezpieczne i przetestowane rozwiązania, eliminując potrzebę tworzenia wszystkiego od podstaw. Dzięki nim można skupić się na logice aplikacji, zamiast tracić czas na implementację zarówno podstawowych funkcji, jak i zaawansowanych rozwiązań, takich jak tworzenie interfejsów graficznych czy skomplikowane obliczenia matematyczne.

Wbudowane biblioteki (m.in *math*, *datetime*) dostarczają podstawowych funkcji, natomiast biblioteki dostępne przez społeczność – publikowane na PyPI – oferują szereg funkcjonalności, od przetwarzania danych (*pandas*, *numpy*) po tworzenie interfejsów graficznych (*tkinter*) czy rysowanie grafiki (*turtle*).

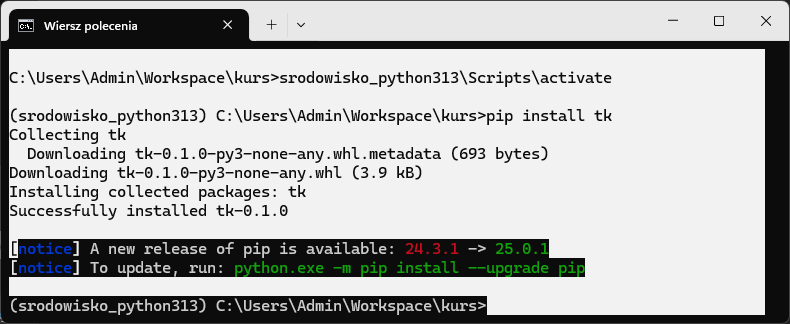
### Instalacja i wykorzystanie bibliotek z pomocą narzędzia PIP

PIP to menadżer 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.



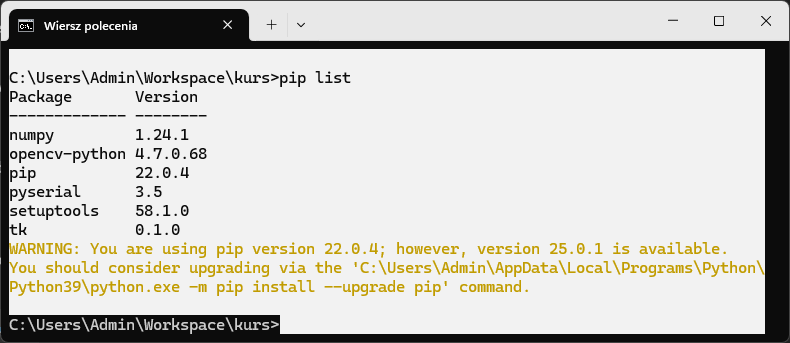
Rysunek 1.46 Przykład instalacji biblioteki tkinter na poziomie globalnym

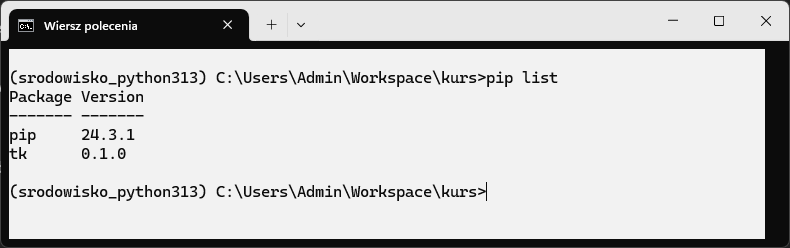
Instalowanie bibliotek za jego pomocą jest bardzo wygodne i szybkie. Należy jednak mieć na uwadze fakt, że wykonywanie go na poziomie systemu, powoduje, że biblioteka jest instalowana globalnie i może to powodować konflikty z innymi modułami korzystającymi z innych zależności. Dlatego też, przy dużych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



Rysunek 1.47 Przykład instalacji biblioteki tkinter wewnątrz środowiska wirtualnego

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





Rysunek 1.48 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.

Listing 1.16. Instalacja biblioteki turtle

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

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

Listing 1.17. Sprawdzenie listy zainstalowanych bibliotek

|  |
| --- |
| pip list |

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

Tabela 1.9. 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.

Listing 1.18. Rysowanie kwadratu za pomocą biblioteki

|  |
| --- |
| [module\_1\_lesson\_4\_turtle\_square.py] |

Listing 1.19. Rysowanie okręgu za pomocą biblioteki

|  |
| --- |
| [module\_1\_lesson\_4\_turtle\_triangle.py] |

### Zadania

**Zadanie 1.8.1. Instalacja dodatkowej biblioteki**

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

**Zadanie 1.8.2. Rysowanie gwiazdy**

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

**Zadanie 1.8.3. Instalacja konkretnej wersji pakietu wewnątrz środowiska wirtualnego**

* Stwórz środowisko wirtualne dla dowolnej wersji języka Python lub aktywuj już stworzone.
* Za pomocą menadżera PIP zainstaluj bibliotekę *matplotlib* w wersji 3.8.2.
* Następnie uaktualnij ją do najnowszej wspieranej wersji.

**Zadanie 1.8.4. \* 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

**Zadanie 1.8.5. \* Rysowanie kwiatu**

* Napisz skrypt, który będzie rysować kwiat za pomocą następującego algorytmu:
  + Ustal liczbę płatków
  + Dla każdego płatka:
    - Narysuj łuk o ustalonym promieniu i kącie
    - Obróć żółwia o kąt równy liczbie kwiatków (podpowiedź: 360 st. / liczba płatków)
  + Po narysowaniu ostatniego płatka, narysuj środek kwiatka będący kołem

## Podsumowanie

Ten rozdział pozwolił zdobyć wiedzę która stanowi fundament operowania w obrębie języka Python. Warto z niego zapamiętać że:

* 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().*

## Zadania podsumowujące

**Zadanie 1.10.1. Instalacja danej wersji Pythona**

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

**Zadanie 1.10.2. Tworzenie skryptu i wyświetlanie wartości w terminalu**

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

**Zadanie 1.10.3. Kalkulator wartości**

Napisz program, który

* Pobiera od użytkownika dwie liczby
* Oblicza i wyświetla ich sumę, różnicę, iloczyn oraz iloraz

**Zadanie 1.10.4. Manipulacja tekstem**

Napisz program, który

* Pobiera od użytkownika imię i nazwisko
* Wyświetla komunikat: „Witaj *[IMIE]* *[NAZWISKO]* !”
  + Imię ma być napisane wielkimi literami, natomiast nazwisko powinno się tylko zaczynać z dużej litery

**Zadanie 1.10.5. Konwersja typów na float i str**

Napis skrypt, w którym

* Zadeklarujesz zmienną i wyświetlisz jej typ
* Przekonwertujesz wartość zmiennej na typ *float* oraz *str* i wyświetlisz typ wynikowy

**Zadanie 1.10.6. Kalkulator jednostek**

Napisz skrypt, który

* Pobiera od użytkownika odległość w kilometrach
* Przelicza ją na wartość w milach i wyświetla wynik

**Zadanie 1.10.7. \* Wyświetlanie sformatowanej daty**

Napisz skrypt, który

* Zaimportuje bibliotekę *datetime*
* Wypisze datę i godzinę w formacie: *miesiąc-dzień-rok godzina:minuta:sekunda*
* Wypisze datę i godzinę w formacie: *dzień/miesiąc/rok godzina:minuta:sekunda*

**Zadanie 1.10.8. Obliczanie obwodu i pola prostokąta**

Napisz skrypt, w którym

* Użytkownik poda długość boku prostokąta
* Program wyświetli obwód i pole prostokąta o podanym boku

**Zadanie 1.10.9. \* 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 1.10.10. \* 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 1.10.11. \* Obliczanie mediany**

Napisz program, który

* Poprosi użytkownika o podanie 7 różnych liczb
* Posortuje liczby w kolejności niemalejącej
* Wyznaczy i wyświetli medianę

**Zadanie 1.10.12. Wyznaczenie sumy zbiorów**

Napisz program, który

* Pobierze od użytkownika dwa zbiory
* Obliczy i wyświetli sumę zbiorów

**Zadanie 1.10.13. Odejmowanie zbiorów**

Napisz program, który

* Pobierze od użytkownika dwa zbiory
* Wyświetli różnicę podanych zbiorów

**Zadanie 1.10.14. Suma rozłączna zbiorów**

Napisz program, który

* Pobierze od użytkownika dwa zbiory
* Wyświetli sumę rozłączną (XOR) podanych zbiorów

**Zadanie 1.10.15 Zliczanie słów w zdaniu**

Napisz program, który

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

**Zadanie 1.10.16 Obliczanie BMI**

Napisz program, który

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

**Zadanie 1.10.17 \* Rysowanie wielokątów**

Napisz program, który

* Pobierze od użytkownika liczbę boków wielokąta
* Za pomocą biblioteki *turtle* narysuj ten wielokąt

**Zadanie 1.10.18 Tworzenie słowników**

Napisz program, który

* Pobierze od użytkownika imię, nazwisko, wiek i zawód
* Wypisze informacje o użytkowniku w postaci słownika

**Zadanie 1.10.19 Tabliczka mnożenia**

Napisz program, który

* Pobierze od użytkownika liczbę od 1 do 10
* Wyświetli wyniki mnożenia tej liczby w zakresie od 1 do 10

**Zadanie 1.10.20 \***

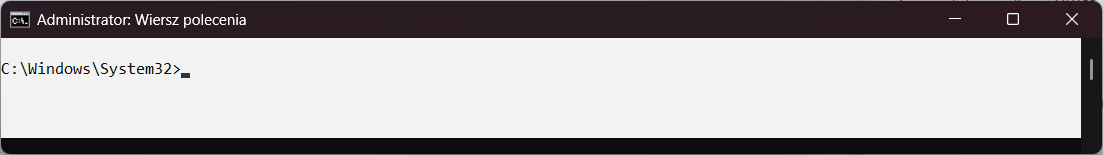
Napisz program, który

* Pobierze od użytkownika dowolną liczbę naturalną
* Za pomocą biblioteki *turtle* narysuje okrąg o zadanym promieniu i wypisze jego obwód i pole

## Dodatek

**Zadanie 1.1.1.**

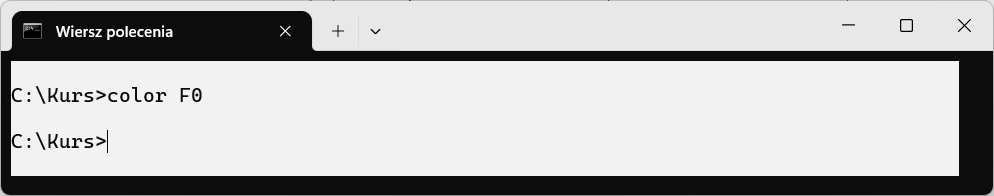
Jedną z metod uruchomienia konsoli z prawami administratora jest wpisanie *CMD* w oknie wyszukiwania systemu Windows, wybranie prawym przyciskiem myszy *Wiersza Poleceń*, a następnie *Uruchom jako administrator*.



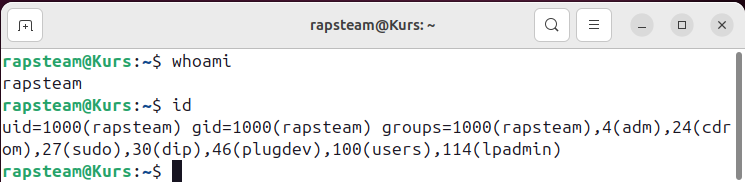
**Zadanie 1.1.2.**

* 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.**

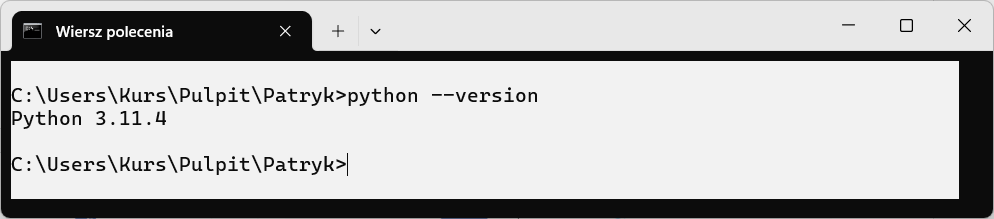
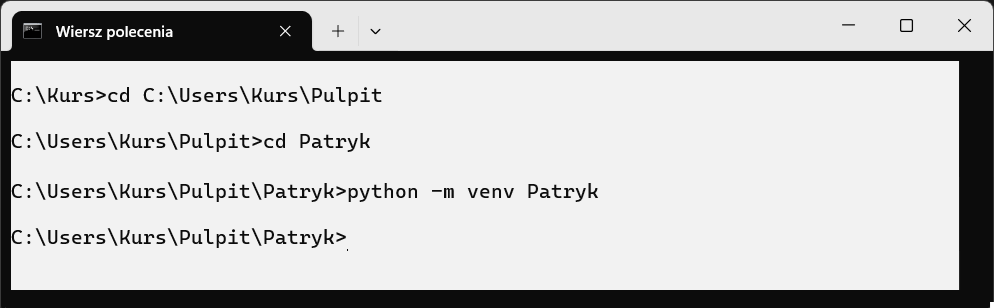
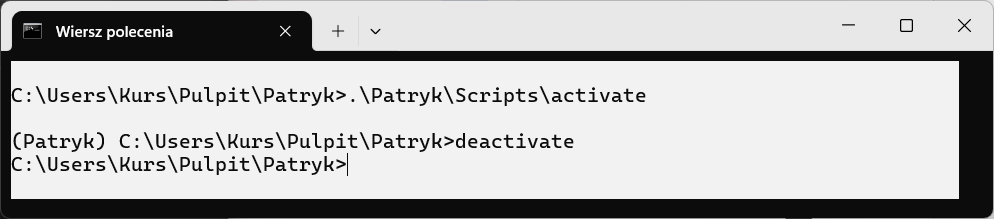


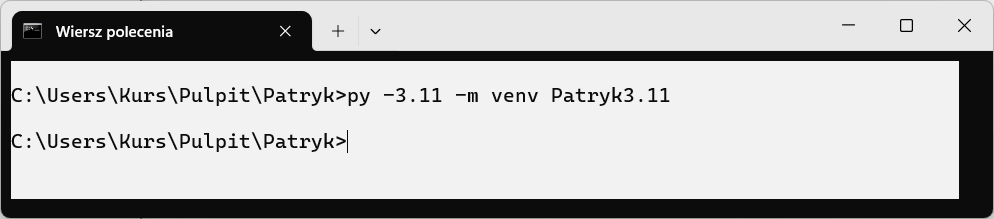
**Zadanie 1.1.4. \***

I

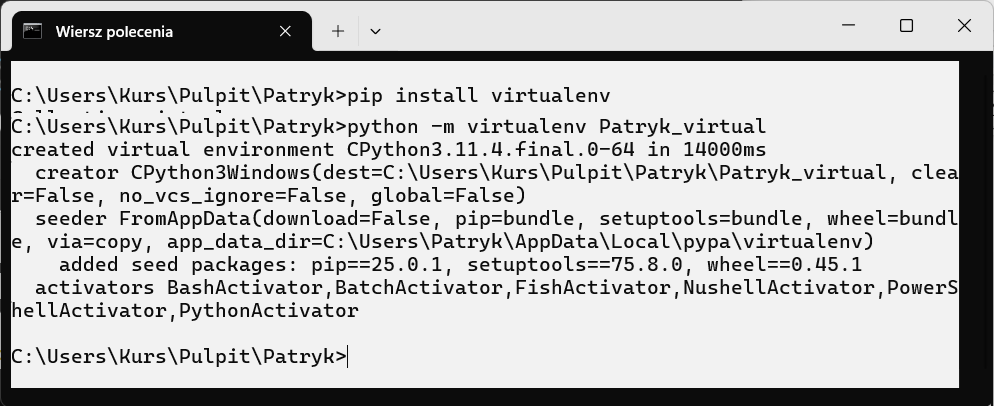
**Zadanie 1.1.5. \***

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

**Zadanie 1.2.1.**   
  
**Zadanie 1.2.2.**  
  
**Zadanie 1.2.3.**  


**Zadanie 1.2.4. \***   


**Zadanie 1.2.5. \***



**Zadanie 1.3.1.**

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

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

**Zadanie 1.3.2.**

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

**Zadanie 1.3.3.**

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

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

**Zadanie 1.3.4. \***

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

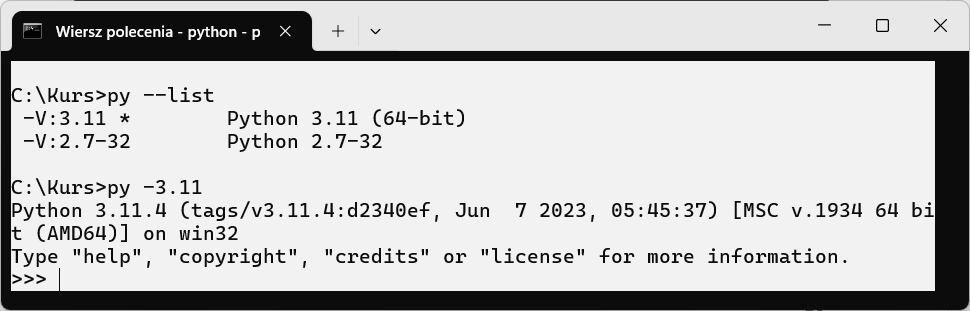
[module\_1\_lesson\_2\_1\_3\_4.png]

**Zadanie 1.3.5. \***

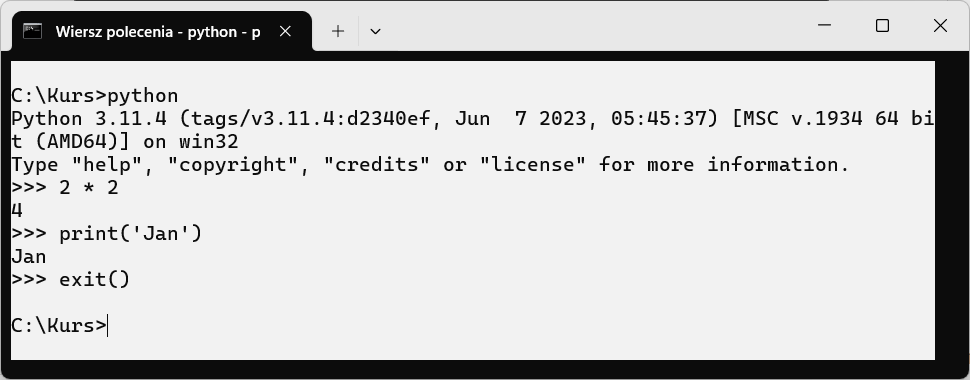
|  |
| --- |
| [module\_1\_lesson\_2\_1\_3\_5.py] |

[module\_1\_lesson\_2\_1\_3\_5.png]

**Zadanie 1.4.1.**



**Zadanie 1.4.2.**



**Zadanie 1.4.3.**

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

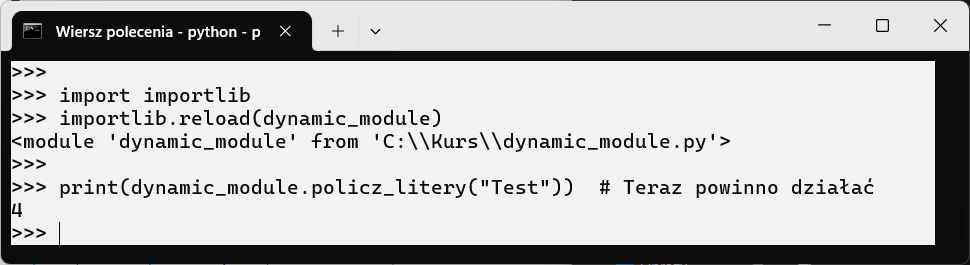
**Zadanie 1.4.4. \***

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

**Zadanie 1.4.5. \***

Funkcja *policz\_litery* zadziała dopiero po ponowny załadowaniu biblioteki poprzez *importlib.reload(dynamic\_module).*

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



**Zadanie 1.5.1.**

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

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

**Zadanie 1.5.2.**

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

[module\_1\_lesson\_3\_1\_5\_2.png]

**Zadanie 1.5.3.**

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

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

**Zadanie 1.5.4.**

|  |
| --- |
| [module\_1\_lesson\_3\_1\_5\_4.py] |

[module\_1\_lesson\_3\_1\_5\_4.png]

**Zadanie 1.5.5.**

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

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

**Zadanie 1.6.1.**

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

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

**Zadanie 1.6.2.**

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

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

**Zadanie 1.6.3.**

|  |
| --- |
| [module\_1\_lesson\_4\_1\_6\_3.py] |

[module\_1\_lesson\_4\_1\_6\_3.png]

**Zadanie 1.6.4.**

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

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

**Zadanie 1.6.5.**

|  |
| --- |
| [module\_1\_lesson\_4\_1\_6\_5.py] |

[module\_1\_lesson\_4\_1\_6\_5.png]

**Zadanie 1.7.1.**

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

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

**Zadanie 1.7.2.**

|  |
| --- |
| [module\_1\_lesson\_4\_1\_7\_2.py] |

[module\_1\_lesson\_4\_1\_7\_2.png]

**Zadanie 1.7.3.**

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

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

**Zadanie 1.7.4.**

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

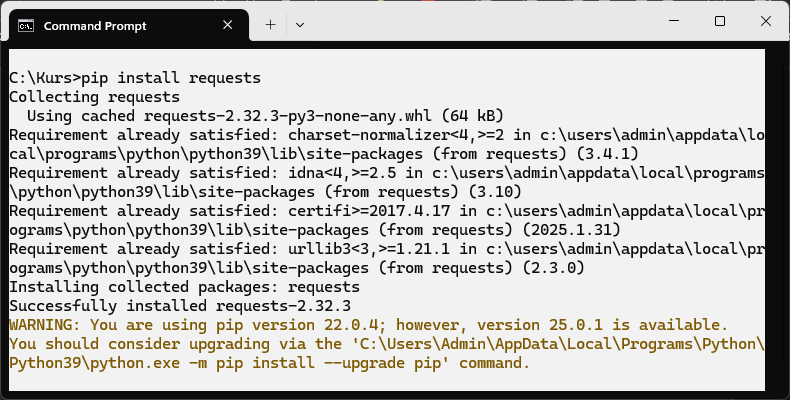
[module\_1\_lesson\_4\_1\_7\_4.png]

**Zadanie 1.7.5.**

|  |
| --- |
| [module\_1\_lesson\_4\_1\_7\_5.py] |

[module\_1\_lesson\_4\_1\_7\_5.png]

**Zadanie 1.8.1.**

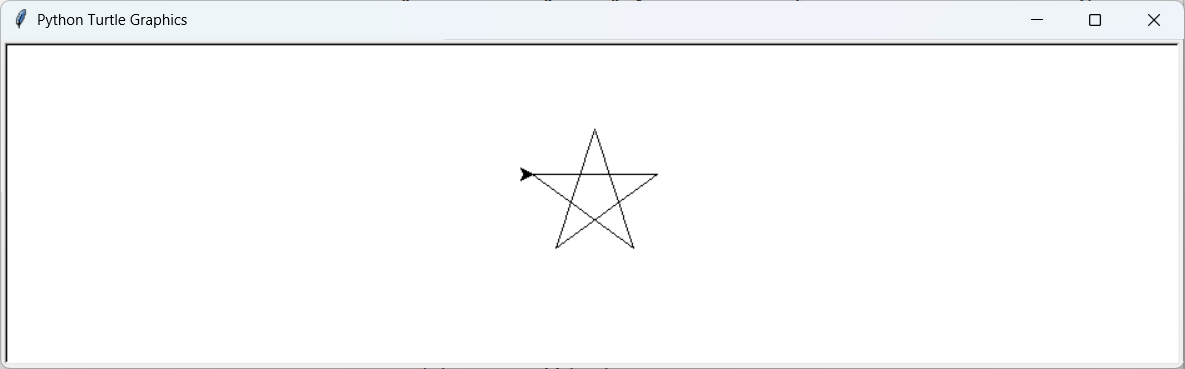


|  |
| --- |
| [module\_1\_lesson\_4\_1\_8\_1.py] |

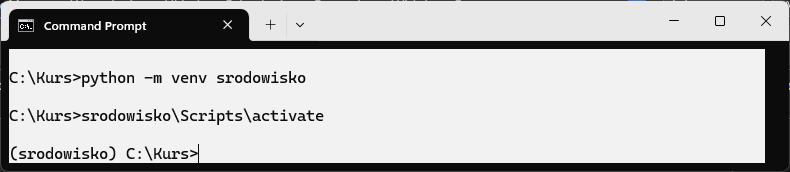
[module\_1\_lesson\_4\_1\_8\_1.png]

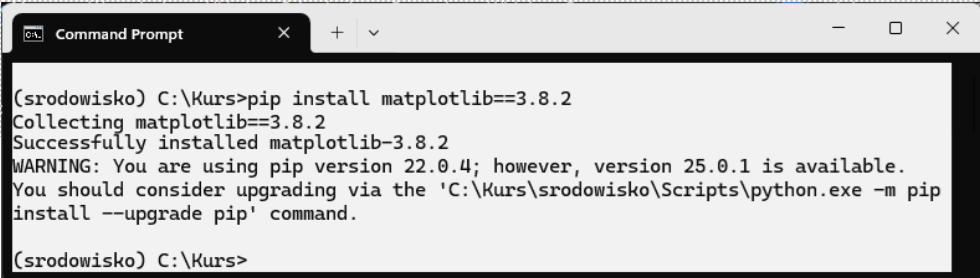
**Zadanie 1.8.2.**

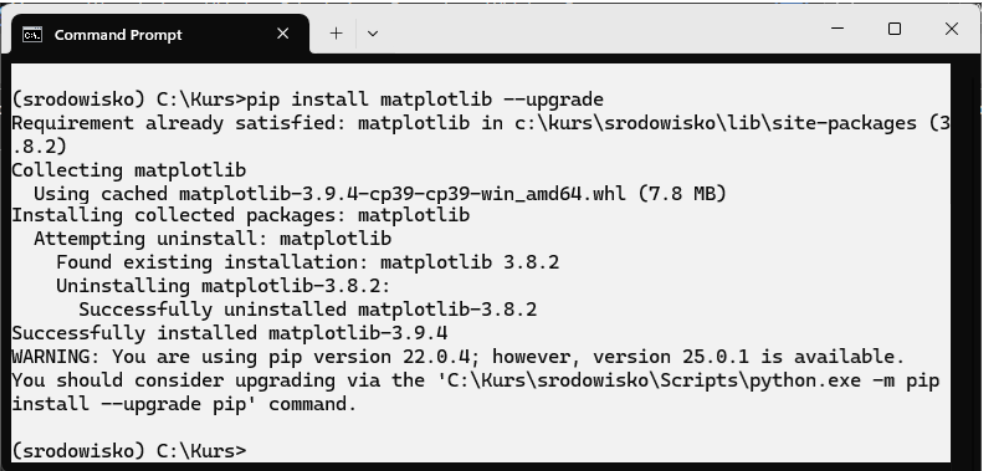
|  |
| --- |
| [module\_1\_lesson\_4\_1\_8\_2.py] |



**Zadanie 1.8.3.**

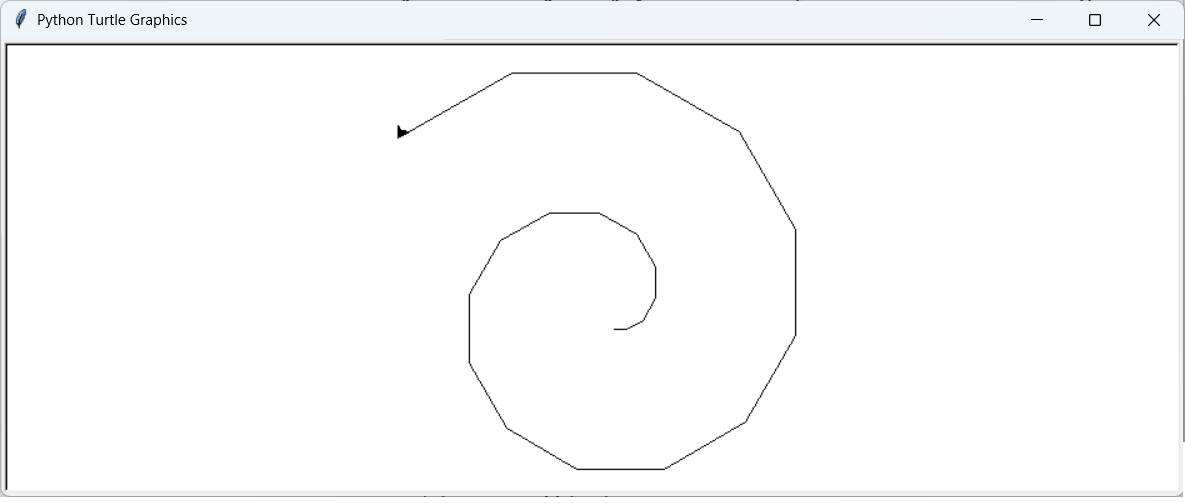






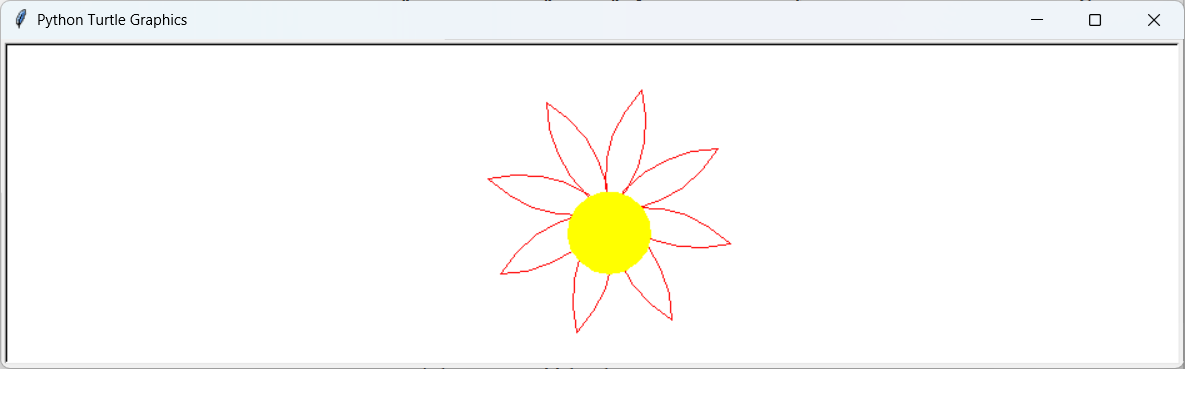
**Zadanie 1.8.4. \***

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

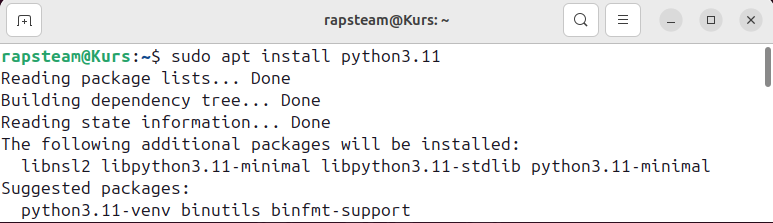


**Zadanie 1.8.5. \***

|  |
| --- |
| [module\_1\_lesson\_4\_1\_8\_5.py] |



**Zadanie 1.10.1.**



**Zadanie 1.10.2.**

|  |
| --- |
| [module\_1\_summary\_1\_10\_2.py] |

[module\_1\_summary\_1\_10\_2.png]

**Zadanie 1.10.3.**

|  |
| --- |
| [module\_1\_summary\_1\_10\_3.py] |

[module\_1\_summary\_1\_10\_3.png]

**Zadanie 1.10.4.**

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

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

**Zadanie 1.10.5.**

|  |
| --- |
| [module\_1\_summary\_1\_10\_5.py] |

[module\_1\_summary\_1\_10\_5.png]

**Zadanie 1.10.6. \***

|  |
| --- |
| [module\_1\_summary\_1\_10\_6.py] |

[module\_1\_summary\_1\_10\_6.png]

**Zadanie 1.10.7. \***

|  |
| --- |
| [module\_1\_summary\_1\_10\_7.py] |

[module\_1\_summary\_1\_10\_7.png]

**Zadanie 1.10.8.**

|  |
| --- |
| [module\_1\_summary\_1\_10\_8.py] |

[module\_1\_summary\_1\_10\_8.png]

**Zadanie 1.10.9. \***

|  |
| --- |
| [module\_1\_summary\_1\_10\_9.py] |

[module\_1\_summary\_1\_10\_9.png]

**Zadanie 1.10.10. \***

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

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

**Zadanie 1.10.11. \***

|  |
| --- |
| [module\_1\_summary\_1\_10\_11.py] |

[module\_1\_summary\_1\_10\_11.png]

**Zadanie 1.10.12.**

|  |
| --- |
| [module\_1\_summary\_1\_10\_12.py] |

[module\_1\_summary\_1\_10\_12.png]

**Zadanie 1.10.13**

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

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

**Zadanie 1.10.14**

|  |
| --- |
| [module\_1\_summary\_1\_10\_14.py] |

[module\_1\_summary\_1\_10\_14.png]

**Zadanie 1.10.15**

|  |
| --- |
| [module\_1\_summary\_1\_10\_15.py] |

[module\_1\_summary\_1\_10\_15.png]

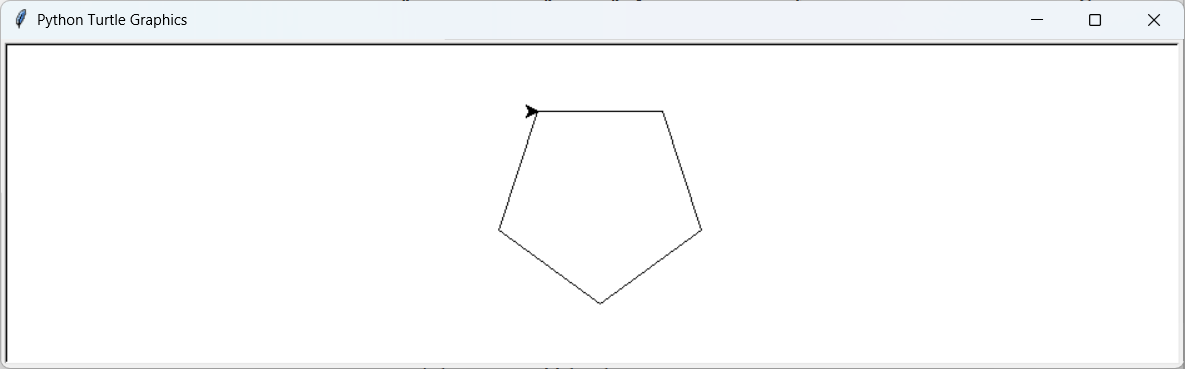
**Zadanie 1.10.16**

|  |
| --- |
| [module\_1\_summary\_1\_10\_16.py] |

[module\_1\_summary\_1\_10\_16.png]

**Zadanie 1.10.17**

|  |
| --- |
| [module\_1\_summary\_1\_10\_17.py] |



**Zadanie 1.10.18**

|  |
| --- |
| [module\_1\_summary\_1\_10\_18.py] |

[module\_1\_summary\_1\_10\_18.png]

**Zadanie 1.10.19**

|  |
| --- |
| [module\_1\_summary\_1\_10\_19.py] |

[module\_1\_summary\_1\_10\_19.png]

**Zadanie 1.10.20**

|  |
| --- |
| [module\_1\_summary\_1\_10\_20.py] |

# Rozpoczynamy kodowanie w języku Python

Po zainstalowaniu Pythona i zapoznaniu się z jego podstawami nadszedł czas, aby zagłębić się w proces programowania. W tym module uczestnicy przygotują swoje środowisko pracy, poznają narzędzia ułatwiające programowanie oraz rozpoczną pisanie pierwszych programów. Omówione zostaną różne środowiska programistyczne, ze szczególnym uwzględnieniem Thonny i Visual Studio Code. Dodatkowo poruszony zostanie temat, dlaczego stosowanie wcięć i spacji jest kluczowe w Pythonie, a na koniec uruchomimy pierwszy program w zintegrowanym środowisku.

## Przygotowanie środowiska IDE

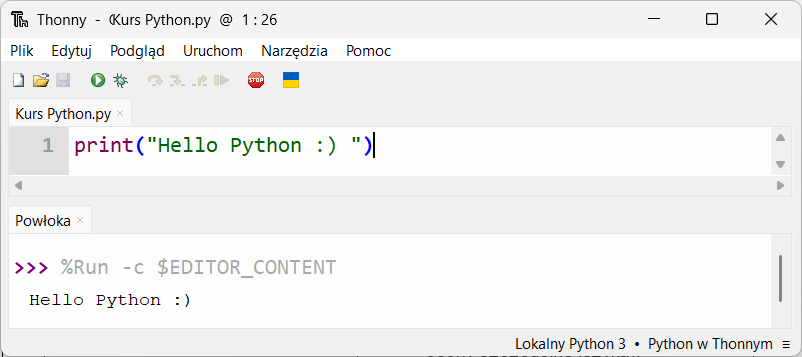
IDE, czyli zintegrowane środowisko programistyczne (ang. Integrated Development Environment), ułatwia tworzenie oprogramowania dzięki połączeniu wielu funkcji, takich jak autoformatowanie, debugowanie i kompilacja, w jednej aplikacji. Dzięki temu nie trzeba ręcznie wykonywać poszczególnych kroków – narzędzie automatycznie obsługuje wiele procesów związanych z pisaniem kodu. Umożliwia to szybkie napisanie i uruchomienie programu bez dodatkowych etapów. Dla języka Python, dostępne jest wiele tego typu rozwiązań, które zostaną przybliżone.

### Omówienie różnych IDE

W module pierwszym 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.

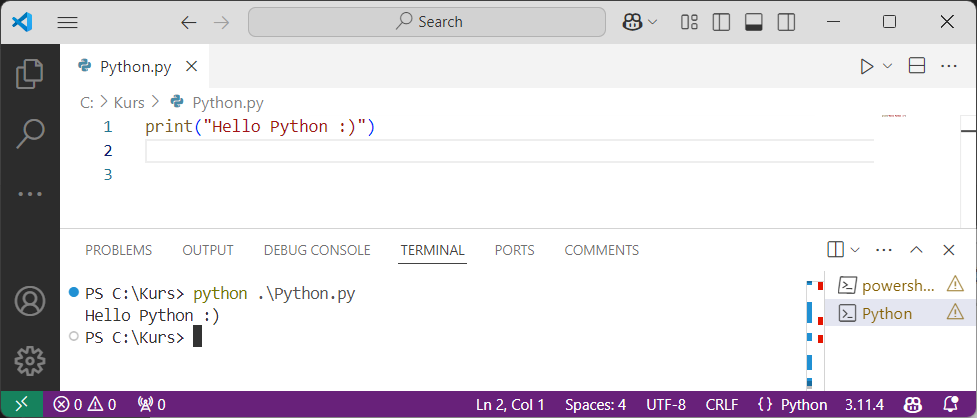
**Thonny**

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.



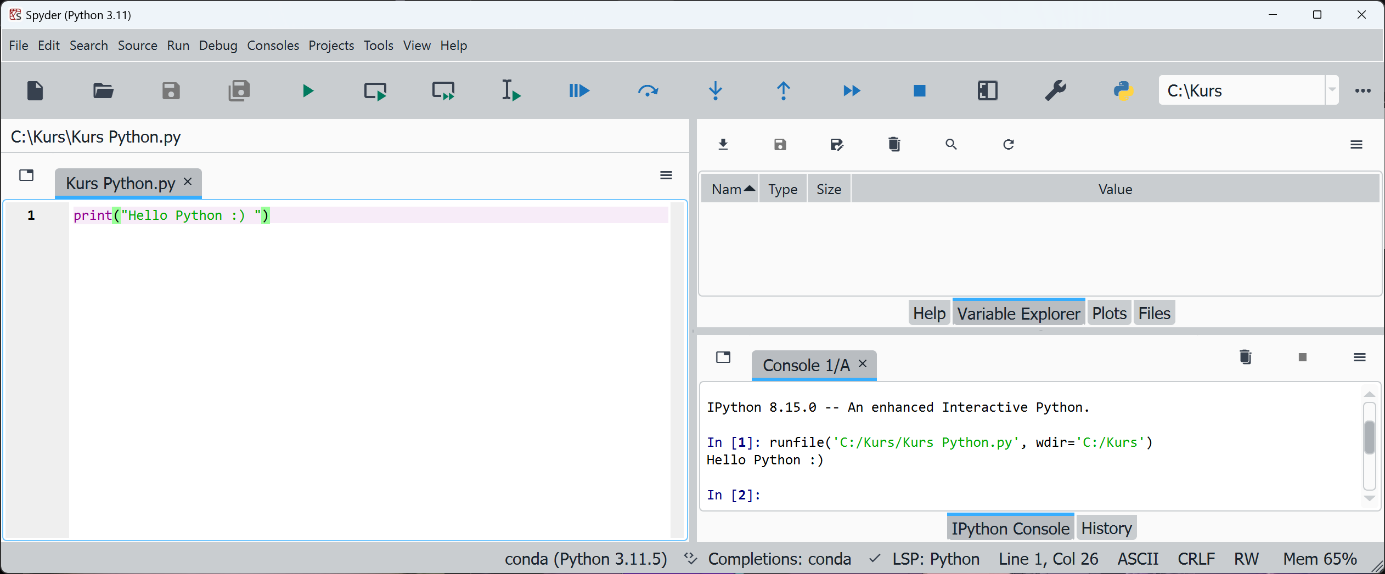
**Visual Studio Code**

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.



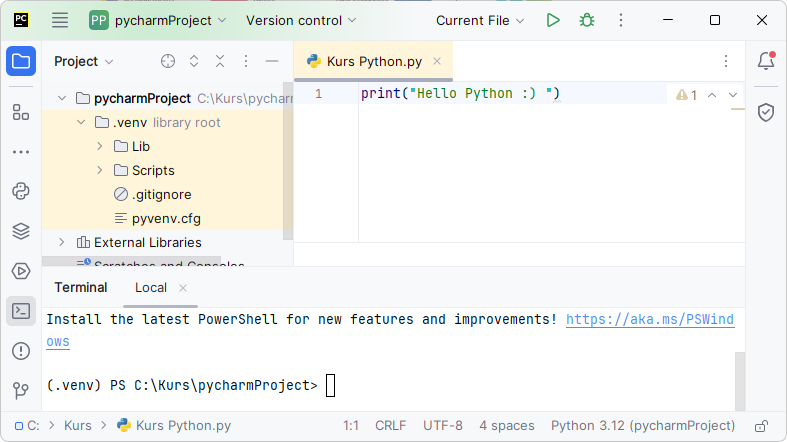
**Spyder**

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.



**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.



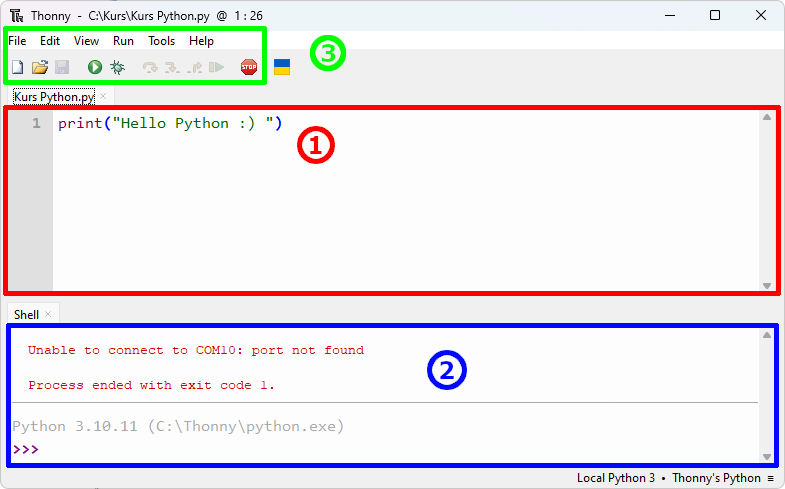
### Instalacja wybranych IDE: Thonny i Visual Studio Code

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.

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

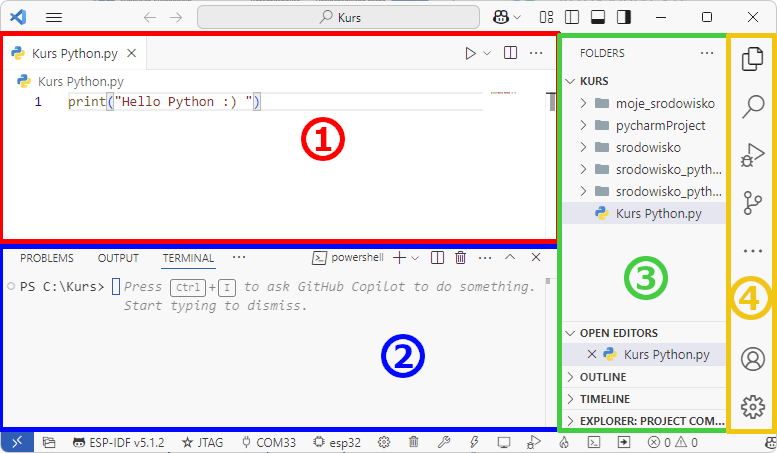
### Zapoznanie się z funkcjami Thonny

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.



### 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.



### Zadania

**Zadanie 2.1.1. Edytor czy IDE**

W 1 zdaniu wyjaśnij różnice pomiędzy edytorem kodu a IDE.

**Zadanie 2.1.2. Instalacja i uruchomienie IDE**

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

**Zadanie 2.1.3. Błąd logiczny w Thonny**

Uruchom Thonny i napisz program, który zawiera błąd logiczny: print(5 / 0)

**Zadanie 2.1.4. \* Weryfikacja instalacji interpretera Python w Thonny**

Otwórz Thonny, a następnie sprawdź, jaki interpreter Python jest aktualnie ustawiony.

**Zadanie 2.1.5. \* 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.

## Zrozumienie zasad wcięć oraz formatowania kodu

Język Python, jak każdy język programistyczny, ma zbiór zasad których należy przestrzegać przy pisaniu kodu. Pozwala to nie tylko na jego bezproblemowe działanie, ale również na uniknięcie błędów, które na pierwszy rzut oka nie powinny występować.

### 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.

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

### Zadania

**Zadanie 2.2.1. Popraw błędy w wcięciach**

|  |
| --- |
| def policz\_sume(a, b):  result = a + b  return result    print(policz\_sume(5, 10)) |

**Zadanie 2.2.2. Popraw błędy w wcięciach**

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

**Zadanie 2.2.3. Popraw formatowanie kodu do PEP 8**

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

**Zadanie 2.2.4. Znajdź i porpaw błąd związany z wcięciem**

|  |
| --- |
| def przywitaj():  print("Witaj!")  print("Miło Cię widzieć.")    przywitaj() |

**Zadanie 2.2.5. Refaktoryzacja kodu zgodnie z PEP 8**

|  |
| --- |
| def oblicz\_cene(ilosc, cena\_za\_sztuke, rabat=0): if(ilosc>10): rabat=0.1 elif ilosc>5:rabat=0.05 else:rabat=0 cena\_koncowa=(ilosccena\_za\_sztuke)(1-rabat) return cena\_koncowa  print( oblicz\_cene(7 , 20)) print( oblicz\_cene(12, 15 ) ) |

## Pierwszy program w IDE

Mając już działające IDE pora przejść z kodowania w edytorze tekstu lub terminalu   
i napisać swój pierwszy kod oraz przeanalizować jak on działa w zintegrowanym środowisku programistycznym.

### 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.

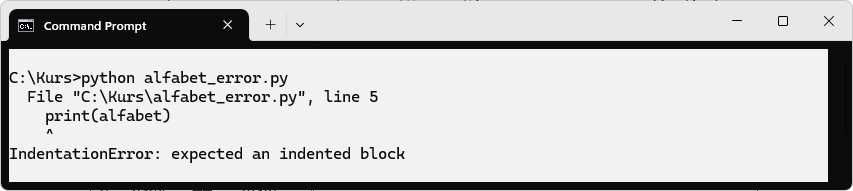
|  |
| --- |
| [module\_2\_lesson\_2\_alfabet.py] |

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

### 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?

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



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 2.3.1. Wypisanie imienia**

Napisz skrypt w IDE który wypisze imię Jan.

**Zadanie 2.3.2. Suma liczb**

Napisz skrypt w IDE który wypisze sumę liczb 5 + 5.

**Zadanie 2.3.3. Błąd w strukturze kodu**

Uruchom poniższy kod w IDE i przeanalizuj, następnie popraw go tak, aby działał poprawnie.

|  |
| --- |
| if name == "main": print("Program startuje...")  alfabet = ["a", "b", "c", "d"]  print(alfabet) |

**Zadanie 2.3.4. 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 2.3.5. \* 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) |

## Zmienne i komentarze

W module pierwszym poruszono temat typów zmiennych jakie występują w języku Python oraz jakie są ich ogólne zastosowania. W poniższym rozdziale w szczególe zostaną przybliżone sposoby deklaracji zmiennych, konwencje nazewnictwa i zakresy ich istnienia a także czym są komentarze w kodzie i dlaczego warto je stosować.

### 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.

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

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

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.

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

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

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. |

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

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

### 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*).

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

Od wersji Pyhona 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.

|  |
| --- |
| [module\_2\_lesson\_2\_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.*

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

### Zadania

**Zadanie 2.4.1. Deklaracja zmiennej typu całkowitego**

Zadeklaruj zmienną typu całkowitego stosując adnotacje *int*,

**Zadanie 2.4.2. Zmiena typu z int na str**

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

**Zadanie 2.4.3. Dekalracja liczy zmiennoprzecinkowej**

Zadeklaruj liczbę zmiennoprzecinkową z adnotacją typu float.

**Zadanie 2.4.4. Komentrzenie pojawiaja sie w kodzie**

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

**Zadanie 2.4.5. Inne typy z bibliotek (numpy)**

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

## Podsumowanie

Z powyższego rozdziału warto zapamiętać, że:

* 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

**Zadanie 2.6.1. Popularne IDE**

* Wyszukaj i podaj trzy popularne IDE do programowania w Pythonie.

**Zadanie 2.6.2. Różnice edytora od IDE**

* Wymień trzy różnice między edytorem kodu a IDE.

**Zadanie 2.6.3. Znaczenie komentrzy**

* Wyjaśnij znaczenie wcięć w języku Python.

**Zadanie 2.6.4. Zasady PEP 8**

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

**Zadanie 2.6.5. Rodzaje komentarzy**

* Jakie są dwa główne sposoby komentowania kodu w Pythonie i w jakich sytuacjach warto ich używać?

**Zadanie 2.6.6. Typy danych**

* Wymień kilka podstawowych typów danych w Pythonie.

**Zadanie 2.6.7. Typy int oraz flaot**

* Czym różni się float od int w Pythonie i kiedy warto używać każdego z tych typów?

**Zadanie 2.6.8. Konwersja typu**

* Co to jest konwersja typu w Pythonie? Podaj przykład zmiany int na str i odwrotnie.

**Zadanie 2.6.9. Nazewnictwo zmiennych**

* Podaj trzy konwencje nazewnictwa zmiennych i wyjaśnij ich znaczenie.

**Zadanie 2.6.10. Trzy zmienne, trzy typy**

* Zadeklaruj imie (str), wiek (int), srednia\_ocen (float).

**Zadanie 2.6.1. Zmiana int na str**

* Zadeklaruj wiek jako int, a następnie zmień go na str.

**Zadanie 2.6.12. Zmienna *float***

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

**Zadanie 2.6.13. Wiele zmiennych w jednej linii**

* Zadeklaruj trzy zmienne (x, y, z) w jednej linii i przypisz im odpowiednie wartości.

**Zadanie 2.6.14. Dynamiczna zmiana typu zmiennej**

* Utwórz zmienną wartosc, przypisz jej liczbę całkowitą, a następnie zmień jej typ na str i wypisz jej typ przed i po zmianie.

**Zadanie 2.6.15. Wspólne przypisanie tej samej wartośc**i

* Zadeklaruj trzy zmienne (a, b, c) i przypisz im tę samą wartość w jednej linii, a następnie wypisz ich wartości.

**Zadanie 2.6.16. Konwersja int na float**

* Zadeklaruj zmienną liczba jako int, a następnie przekonwertuj ją na float.

**Zadanie 2.6.17. Zmienna logiczna**

* Zadeklaruj zmienną bool i przekonwertuj na int.

**Zadanie 2.6.18. Dodawanie int i float**

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

**Zadanie 2.6.19. Typ uint16 zumpy**

* Użyj biblioteki numpy, aby stworzyć liczbę uint16 i wypisz jej wartość oraz typ.

**Zadanie 2.6.20. Konwersja None**

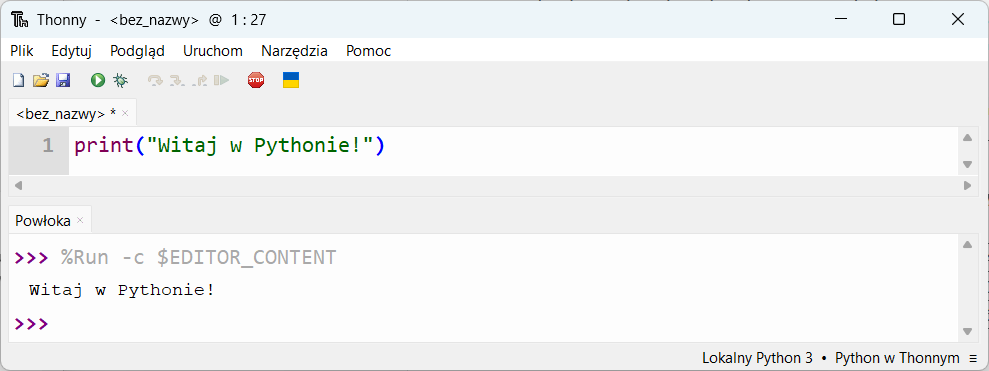
* Stwórz zmienną liczba, która przechowuje wartość None, a następnie zmień ją na int i wypisz wynik.

## Dodatek

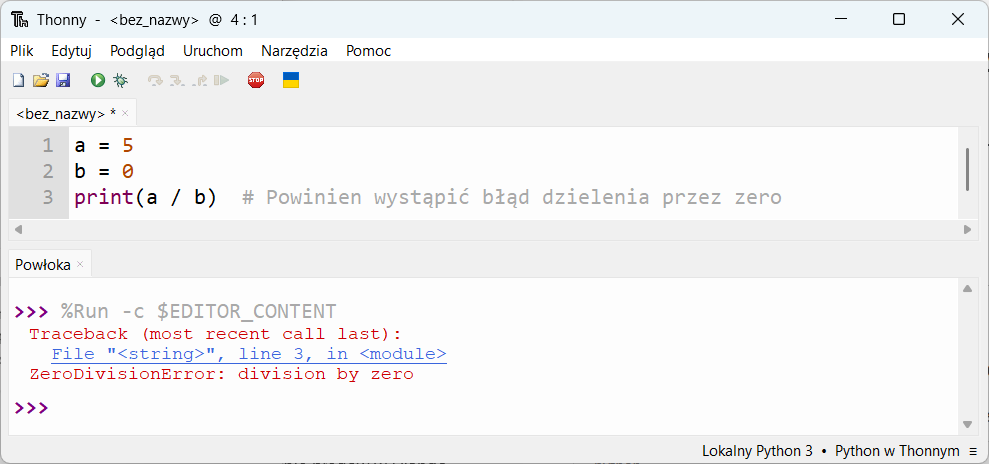
**Zadanie 2.1.1.**

Edytor kodu to lekka aplikacja do pisania i edytowania kodu, natomiast **IDE** (Integrated Development Environment) to bardziej rozbudowane środowisko, które oprócz edytora oferuje dodatkowe narzędzia, takie jak debugowanie, kompilowanie i zarządzanie projektami.

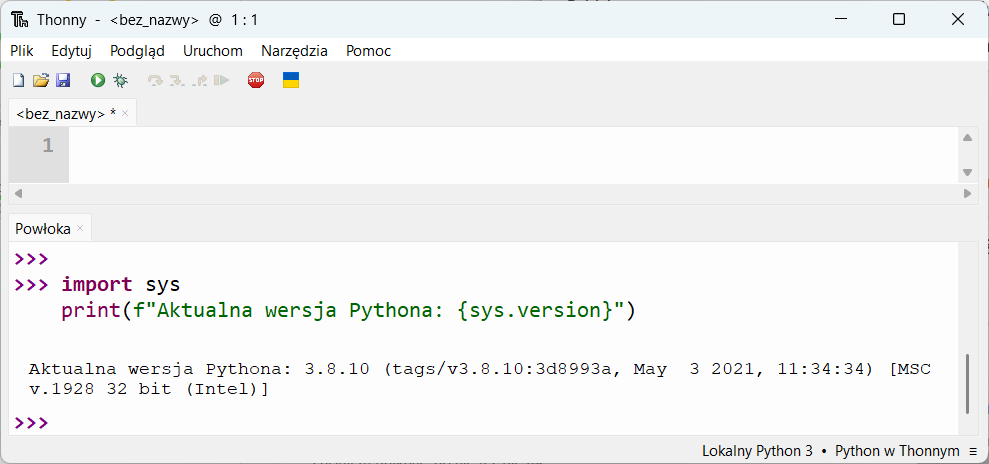
**Zadanie 2.1.2.**



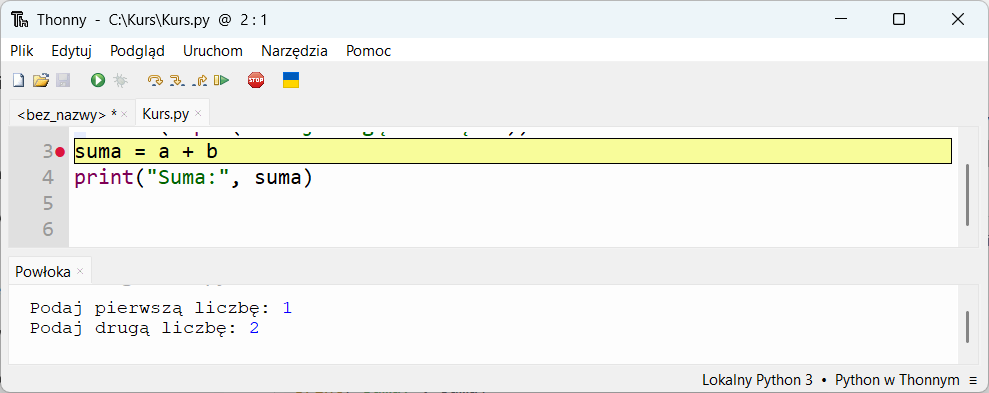
**Zadanie 2.1.3.**



**Zadanie 2.1.4.**



**Zadanie 2.1.5.**



**Zadanie 2.2.1.**

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

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

**Zadanie 2.2.2.**

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

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

**Zadanie 2.2.3.**

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

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

**Zadanie 2.2.4.**

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

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

**Zadanie 2.2.5.**

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

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

**Zadanie 2.3.1.**

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

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

**Zadanie 2.3.2.**

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

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

**Zadanie 2.3.3.**

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

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

**Zadanie 2.3.4.**

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

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

**Zadanie 2.3.5.**

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

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

**Zadanie 2.4.1.**

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

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

**Zadanie 2.4.2.**

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

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

**Zadanie 2.4.3.**

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

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

**Zadanie 2.4.4.**

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

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

**Zadanie 2.4.5.**

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

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

**Zadanie 2.6.1.**

Trzy popularne IDE do programowania w Pythonie to:

PyCharm – zaawansowane środowisko dla profesjonalistów z wieloma funkcjami.

Visual Studio Code (VS Code) – zaawansowany edytor kodu, ale dzięki rozszerzeniom może pełnić funkcję lekkiego IDE.

Thonny – proste środowisko idealne dla początkujących.

**Zadanie 2.6.2.**

Edytor kodu (np. Notepad++, Sublime Text) to prosty program do pisania kodu, bez dodatkowych funkcji.

IDE (np. PyCharm, VS Code) zawiera dodatkowe narzędzia, takie jak debugger, system kontroli wersji i podpowiedzi składniowe.

IDE zazwyczaj obsługuje wiele języków i pozwala na integrację z zewnętrznymi bibliotekami, edytor kodu jest bardziej podstawowy.

**Zadanie 2.6.3.**

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 2.6.4.**

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 2.6.5.**

Komentarze jednoliniowe, komentarze wieloliniowe.

**Zadanie 2.6.6.**

Typ *int* – liczby całkowite (5, -3)

Typ *float* – liczby zmiennoprzecinkowe (3.14, -0.5)

Typ *str* – łańcuchy znaków ("Python", 'Hello')

Typ *bool* – wartości logiczne (True, False)

Typ *list* – lista ([1, 2, 3])

**Zadanie 2.6.7.**

Typ *int* przechowuje liczby całkowite, np. 5, -10.

Typ *float* przechowuje liczby zmiennoprzecinkowe, np. 3.14, -2.5.

Typ *float* jest potrzebny, gdy pracujemy z wartościami dziesiętnymi (m.in współrzędne, wymiary).

**Zadanie 2.6.8.**

Konwersja typu to proces zmiany typu danych zmiennej w Pythonie. Konwersja może występować jawnie lub niejawnie. Przykład konwersji jawnej to *liczba = int("5”).*

**Zadanie 2.6.9.**

Nazewnictwo *snake\_case* – moja\_zmienna (zalecane w Pythonie).  
Nazewnictwo *CamelCase* – MojaZmienna (częściej stosowane w JavaScript).   
Nazewnictwo *SCREAMING\_SNAKE\_CASE* – ZMIENNA\_KONSTANTA (dla wartosći stałych).

**Zadanie 2.6.10.**

|  |
| --- |
| [module\_2\_summary\_2\_6\_10.py] |

[module\_2\_summary\_2\_6\_10.png]

**Zadanie 2.6.11.**

|  |
| --- |
| [module\_2\_summary\_2\_6\_11.py] |

[module\_2\_summary\_2\_6\_11.png]

**Zadanie 2.6.12.**

|  |
| --- |
| [module\_2\_summary\_2\_6\_12.py] |

[module\_2\_summary\_2\_6\_12.png]

**Zadanie 2.6.13.**

|  |
| --- |
| [module\_2\_summary\_2\_6\_13.py] |

[module\_2\_summary\_2\_6\_13.png]

**Zadanie 2.6.14.**

|  |
| --- |
| [module\_2\_summary\_2\_6\_14.py] |

[module\_2\_summary\_2\_6\_14.png]

**Zadanie 2.6.15.**

|  |
| --- |
| [module\_2\_summary\_2\_6\_15.py] |

[module\_2\_summary\_2\_6\_15.png]

**Zadanie 2.6.16.**

|  |
| --- |
| [module\_2\_summary\_2\_6\_16.py] |

[module\_2\_summary\_2\_6\_16.png]

**Zadanie 2.6.17.**

|  |
| --- |
| [module\_2\_summary\_2\_6\_17.py] |

[module\_2\_summary\_2\_6\_17.png]

**Zadanie 2.6.18.**

|  |
| --- |
| [module\_2\_summary\_2\_6\_18.py] |

[module\_2\_summary\_2\_6\_18.png]

**Zadanie 2.6.19.**

|  |
| --- |
| [module\_2\_summary\_2\_6\_19.py] |

[module\_2\_summary\_2\_6\_19.png]

**Zadanie 2.6.20.**

|  |
| --- |
| [module\_2\_summary\_2\_6\_20.py] |

[module\_2\_summary\_2\_6\_20.png]

# Logika warunkowa i kontrola przepływu programu w kodzie źródłowym

Rozdział dostarcza wiedzy niezbędnej do budowania i analizowania warunków logicznych w programowaniu. Uczestnicy zapoznają się z operatorami logicznymi i **relacyjnymi**, a także nauczą się wykorzystywać instrukcje warunkowe, takie jak if, elif i else, do sterowania przepływem programu. Dzięki zdobytym umiejętnościom będą mogli implementować złożone warunki logiczne oraz dokonywać konwersji typów danych.

## 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.

Tabela operatorów

|  |  |  |
| --- | --- | --- |
| **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.

Listing X: Demonstracja operatorów porównania w języku Python

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

### Zadania

**Zadanie 3.1.1. Porównanie dwóch liczb**

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

**Zadanie 3.1.2. Sprawdzenie parzystości**

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

**Zadanie 3.1.3. 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 3.1.4. Negacja wartości logicznych**

* Zadeklaruj zmienną status z wartością *True* lub *False*, a następnie użyj operatora not, aby odwrócić jej wartość i wypisać wynik.

**Zadanie 3.1.5. \* Sprawdzenie poprawności hasła**

* Napisz program, który sprawdzi, czy podane hasło spełnia następujące warunki:
  + Ma co najmniej 8 znaków
  + Zawiera przynajmniej jedną cyfrę
* Zawiera przynajmniej jedną wielką literę
* Zwróć *True*, jeśli hasło jest poprawne lub *False* w przeciwnym przypadku, bez użycia *if*.

## 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.

|  |
| --- |
| [module\_3\_lesson\_2\_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.

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

### Zadania

**Zadanie 3.2.1. Sprawdzenie przedziału za pomocą *if***

* Napisz instrukcję if, która sprawdza, czy liczba x jest większa od 10 i mniejsza od 50.

**Zadanie 3.2.2. Liczba parzysta lub większa niż 100**

* Sprawdź, czy podana liczba *y* jest parzysta lub większa od 100. Jeśli którykolwiek z tych warunków jest spełniony, program powinien to wykryć.

**Zadanie 3.2.3. Liczba dodatnia i różna od zera**

* Liczba z nie może być równa zero i musi być dodatnia. Stwórz odpowiedni warunek *if*, który zweryfikuje te dwa warunki jednocześnie.

**Zadanie 3.2.4. 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 3.2.5. 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.

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

### Zadania

**Zadanie 3.3.1. Analiza przedziałów liczbowych za pomocą *in***

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

**Zadanie 3.3.2. 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 3.3.3. Sprawdzenie liczby całkowitej**

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

**Zadanie 3.3.4. \* 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 3.3.5. \* Walidacja danych użytkownika**

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

## Konwersje typów

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.

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

### Zadania

**Zadanie 3.4.1. Konwersja stringa na liczbę**

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

**Zadanie 3.4.2. 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 3.4.3. Niejawna konwersja typów**

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

**Zadanie 3.4.4. \* 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 3.4.5. \* 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.

## Podsumowanie

Logika warunkowa pozwala programowi podejmować decyzje na podstawie określonych warunków. Python wykorzystuje do tego instrukcje *if*, *elif*, *else*, wspierane przez operatory relacyjne (==, !=, >, <) i logiczne (*and*, *or*, *not*). Dzięki ich kombinacji można tworzyć złożone warunki decyzyjne.

Zaawansowane techniki, takie jak zagnieżdżone instrukcje warunkowe, operator warunkowy (*x* *if* warunek *else y*) oraz funkcje *any()* i *all()*, ułatwiają analizę wielu wartości jednocześnie. Są one użyteczne w operacjach na danych, np. weryfikacji list liczb czy kontroli dostępu użytkowników.

Ostatnia część rozdziału dotyczy konwersji typów, czyli zmiany jednej struktury danych na inną. Python umożliwia konwersję jawną (*int()*, *float()*, *str()*) oraz niejawne przekształcenia, np. podczas operacji *int* z *float*. Umiejętność przekształcania typów jest przydatna w pracy z danymi pochodzącymi z różnych źródeł.

## Zadania

**Zadanie 3.6.1. Sprawdzenie liczby i jej typu**

* Pobierz wartość od użytkownika i sprawdź, czy jest liczbą całkowitą, zmiennoprzecinkową czy tekstem.

**Zadanie 3.6.2. Konwersja jednostek długości**

* Pobierz wartość w centymetrach i przelicz ją na metry oraz milimetry.

**Zadanie 3.6.3. Przekształcanie tekstu**

* Pobierz tekst, skonwertuj go na wielkie litery, sprawdź, czy składa się tylko z liter, a następnie policz jego długość.

**Zadanie 3.6.4. Konwersja czasu**

* Pobierz liczbę sekund i przelicz ją na godziny, minuty i sekundy.

**Zadanie 3.6.5. 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 3.6.6. 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 3.6.7. Dynamiczne przeliczanie walut**

* Pobierz od użytkownika kwotę w złotówkach i przelicz ją na dolary lub euro w zależności od wyboru, po kursach: dolar = 3,97 zł, euro = 4,16 zł."

**Zadanie 3.6.8. Sprawdzenie, czy podana wartość jest liczbą całkowitą ujemną**

* Pobierz wartość od użytkownika i sprawdź, czy jest liczbą całkowitą i ujemną*.*

**Zadanie 3.6.9. Sprawdzenie poprawności adresu e-mail**

* Pobierz adres e-mail i sprawdź, czy zawiera znak @ oraz ‘.’ w odpowiednich miejscach.

**Zadanie 3.6.10. Sprawdzenie poprawności kodu pocztowego**

* Pobierz kod pocztowy w formacie XX-XXX i zweryfikuj, czy zawiera pięć cyfr i myślnik we właściwym miejscu.

**Zadanie 3.6.11. Zamiana listy liczb na krotkę**

* Pobierz od użytkownika listę cyfr oddzielonych spacją, przekonwertuj je na listę, a następnie zamień na krotkę i wyświetl wynik.

**Zadanie 3.6.12. Data urodzenia i płeć z PESEL**

* Pobierz numer PESEL od użytkownika i sprawdź, czy składa się dokładnie z 11 cyfr, a następnie ustal datę urodzenia i płeć.

**Zadanie 3.6.13. 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 3.6.14. Sprawdzenie czy rok jest przestępny**

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

**Zadanie 3.6.15. Klasyfikacja temperatury**

* Pobierz temperaturę od użytkownika i określ, czy jest niska (poniżej 10°C), umiarkowana (10-25°C) czy wysoka (powyżej 25°C).

**Zadanie 3.6.16. Sprawdzenie wieku i uprawnień do prowadzenia pojazdów**

* Pobierz wiek użytkownika i sprawdź, czy jest wystarczająco stary, aby prowadzić rower (10+ lat), motorower (14+ lat), samochód (18+ lat).

**Zadanie 3.6.17. Automatyczna klasyfikacja masy ciała**

* Pobierz wagę i wzrost użytkownika, oblicz jego BMI i przypisz do kategorii: niedowaga, norma, nadwaga, otyłość.

**Zadanie 3.6.18. Sprawdzenie poprawności identyfikatora VAT UE**

* Pobierz identyfikator VAT UE i sprawdź, czy zaczyna się od poprawnego kodu kraju oraz czy zawiera tylko cyfry i litery.

**Zadanie 3.6.19.** Konwersja jednostek masy

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

**Zadanie 3.6.20. 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 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 3.4.1.**

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

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

**Zadanie 3.4.2.**

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

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

**Zadanie 3.4.3.**

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

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

**Zadanie 3.4.4.**

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

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

**Zadanie 3.4.5.**

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

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

**Zadanie 3.6.1.**

|  |
| --- |
| [module\_3\_summary\_3\_6\_1.py] |

[module\_3\_summary\_3\_6\_1.png]

**Zadanie 3.6.2.**

|  |
| --- |
| [module\_3\_summary\_3\_6\_2.py] |

[module\_3\_summary\_3\_6\_2.png]

**Zadanie 3.6.3.**

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

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

**Zadanie 3.6.4.**

|  |
| --- |
| [module\_3\_summary\_3\_6\_4.py] |

[module\_3\_summary\_3\_6\_4.png]

**Zadanie 3.6.5.**

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

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

**Zadanie 3.6.6.**

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

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

**Zadanie 3.6.7.**

|  |
| --- |
| [module\_3\_summary\_3\_6\_7.py] |

[module\_3\_summary\_3\_6\_7.png]

**Zadanie 3.6.8.**

|  |
| --- |
| [module\_3\_summary\_3\_6\_8.py] |

[module\_3\_summary\_3\_6\_8.png]

**Zadanie 3.6.9.**

|  |
| --- |
| [module\_3\_summary\_3\_6\_9.py] |

[module\_3\_summary\_3\_6\_9.png]

**Zadanie 3.6.10.**

|  |
| --- |
| [module\_3\_summary\_3\_6\_10.py] |

[module\_3\_summary\_3\_6\_10.png]

**Zadanie 3.6.11.**

|  |
| --- |
| [module\_3\_summary\_3\_6\_11.py] |

[module\_3\_summary\_3\_6\_11.png]

**Zadanie 3.6.12.**

|  |
| --- |
| [module\_3\_summary\_3\_6\_12.py] |

[module\_3\_summary\_3\_6\_12.png]

**Zadanie 3.6.13.**

|  |
| --- |
| [module\_3\_summary\_3\_6\_13.py] |

[module\_3\_summary\_3\_6\_13.png]

**Zadanie 3.6.14.**

|  |
| --- |
| [module\_3\_summary\_3\_6\_14.py] |

[module\_3\_summary\_3\_6\_14.png]

**Zadanie 3.6.15.**

|  |
| --- |
| [module\_3\_summary\_3\_6\_15.py] |

[module\_3\_summary\_3\_6\_15.png]

**Zadanie 3.6.16.**

|  |
| --- |
| [module\_3\_summary\_3\_6\_16.py] |

[module\_3\_summary\_3\_6\_16.png]

**Zadanie 3.6.17.**

|  |
| --- |
| [module\_3\_summary\_3\_6\_17.py] |

[module\_3\_summary\_3\_6\_17.png]

**Zadanie 3.6.18.**

|  |
| --- |
| [module\_3\_summary\_3\_6\_18.py] |

[module\_3\_summary\_3\_6\_18.png]

**Zadanie 3.6.19.**

|  |
| --- |
| [module\_3\_summary\_3\_6\_19.py] |

[module\_3\_summary\_3\_6\_19.png]

**Zadanie 3.6.20.**

|  |
| --- |
| [module\_3\_summary\_3\_6\_20.py] |

[module\_3\_summary\_3\_6\_20.png]

# Napisy i działania matematyczne

Każdy program komputerowy to w istocie przetwarzanie danych – mogą to być liczby, teksty czy bardziej złożone struktury informacji. W codziennym życiu spotykamy się z nimi na każdym kroku: od prostych wiadomości tekstowych po zaawansowane obliczenia w aplikacjach. W tym rozdziale przyjrzymy się podstawowym operacjom na łańcuchach znaków, liczbach oraz strukturom danych, takim jak listy, krotki i słowniki.

## 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 i to długość napisu.

|  |
| --- |
| [module\_4\_lesson\_1\_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\_4\_lesson\_1\_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\_4\_lesson\_1\_strings\_formatting.py] |

### Zadania

**Zadanie 4.1.1. Imię z wielkiej litery**

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

**Zadanie 4.1.2. Łączenie danych za pomocą f-string**

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

**Zadanie 4.1.3. Formatowanie długości napisu**

* Wczytaj dowolny tekst i wyświetl jego długość w zdaniu.

**Zadanie 4.1.4. Wyświetlanie danych w kolumnach**

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

**Zadanie 4.1.5. \* Formatowanie procentów**

* Wczytaj liczbę i wyświetl ją w formacie procentowym.

## Działania matematyczne i liczby

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.

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

### Zadania

**Zadanie 4.2.1. Typy liczb**

* Wczytaj liczbę całkowitą oraz zmiennoprzecinkową i wypisz ich typy

**Zadanie 4.2.2. Zaokrąglanie**

* Zaokrąglij liczbę do 3 miejsc

**Zadanie 4.2.3. Formatowanie liczby w komunikacie**

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

**Zadanie 4.2.4. Operacje matematyczne**

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

**Zadanie 4.2.5. \* Formatowanie procentów**

* Wczytaj liczbę i wyświetl ją w formacie procentowym.

## 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\_4\_lesson\_3\_lists.py] |

**Kortki**

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\_4\_lesson\_3\_tuples.py] |

**Słowinki**

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

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

### Zadania

**Zadanie 4.3.1. Lista owoców**

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

**Zadanie 4.3.2. Modyfikacja listy liczb**

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

**Zadanie 4.3.3. Krotka z danymi użytkownika**

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

**Zadanie 4.3.4.** **Słownik z danymi samochodu**

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

**Zadanie 4.3.5. \*** Iteracja po słowniku

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

## Podsumowanie

W tym rozdziale poznaliśmy podstawowe operacje na napisach, liczbach oraz strukturach danych, takich jak listy, krotki i słowniki. Omówiliśmy sposoby manipulowania tekstem, formatowania ciągów znaków oraz pobierania danych od użytkownika. Zapoznaliśmy się również z operacjami matematycznymi, funkcjami liczbowymi oraz metodami zaokrąglania i formatowania wyników.

Następnie przyjrzeliśmy się listom i krotkom jako strukturom przechowującym sekwencje danych oraz słownikom, które organizują informacje w pary klucz-wartość.

## Zadania

**Zadanie 4.5.1. Zamiana liter**

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

**Zadanie 4.5.2. Sprawdzanie końcówki**

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

**Zadanie 4.5.3. Liczenie znaków**

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

**Zadanie 4.5.4. Skracanie tekstu**

* Wczytaj tekst i skróć go do pierwszych 10 znaków.

**Zadanie 4.5.5. Odwracanie tekstu**

* Wczytaj tekst i wyświetl go w odwrotnej kolejności.

**Zadanie 4.5.6. 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 4.5.7. Dzielenie całkowite**

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

**Zadanie 4.5.8. Obliczanie kwadratu i pierwiastka**

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

**Zadanie 4.5.9.** **Liczba do potęgi**

* Wczytaj liczbę i podnieś ją do potęgi drugiej.

**Zadanie 4.5.10. Obliczanie średniej**

* Wczytaj trzy liczby i oblicz ich średnią arytmetyczną.

**Zadanie 4.5.11.** **Dodawanie elementów do listy**

* Wczytaj trzy słowa i dodaj je do listy, a następnie wyświetl listę.

**Zadanie 4.5.12. Znajdowanie największej liczby**

* Wczytaj pięć liczb do listy i znajdź największą wartość.

**Zadanie 4.5.13. Usuwanie elementu**

* Wczytaj listę pięciu liczb i usuń pierwszy element.

**Zadanie 4.5.14. Sortowanie listy**

* Wczytaj pięć liczb i posortuj je w kolejności rosnącej.

**Zadanie 4.5.15. Suma elementów listy**

* Wczytaj pięć liczb i oblicz ich sumę.

**Zadanie 4.5.16. Tworzenie krotki**

* Wczytaj imię, wiek i ulubiony kolor, a następnie zapisz je w krotce.

**Zadanie 4.5.17. Odczyt z krotki**

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

**Zadanie 4.5.18. Dodawanie danych do słownika**

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

**Zadanie 4.5.19. Modyfikacja wartości w słowniku**

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

**Zadanie 4.5.20. Iteracja po słowniku**

* Stwórz słownik zawierający trzy produkty i ich ceny, a następnie wyświetl każdy produkt i jego cenę w osobnej linii.

## 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 4.3.1.**

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

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

**Zadanie 4.3.2.**

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

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

**Zadanie 4.3.3.**

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

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

**Zadanie 4.3.4.**

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

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

**Zadanie 4.3.5.**

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

[module\_4\_lesson\_3\_4\_3\_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]

# Funkcje i pętle

Funkcje i pętle to podstawowe narzędzia, które umożliwiają tworzenie bardziej uporządkowanego kodu. Funkcje pozwalają na grupowanie powiązanych ze sobą instrukcji w jeden blok, który można wielokrotnie wywoływać w różnych miejscach programu. Dzięki temu można uniknąć powtarzania tego samego kodu, a całość staje się bardziej czytelna i łatwiejsza w utrzymaniu.

Pętle natomiast 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.

W tym rozdziale dowiesz się, jak definiować funkcje, przekazywać do nich argumenty i zwracać wyniki, a także jak stosować pętle, aby iterować po strukturach danych i wykonywać powtarzalne zadania.

## Tworzenie funkcji w języku Python

Funkcja to jedno z najważniejszych narzędzi jakie występują w języku Python. Może to ilustrować fakt, że istnieje cały paradygmat programowania, nazywany programowaniem funkcyjnym. Funkcja składa się z nazwy i ciała deklarowanego po wcięciu. Przyjmuje ona argumenty, czyli zmienne które w swoim ciele przetwarza i może te przetworzone wartości zwracać. Tworząc i korzystając z nich należy pamiętać, że taka deklaracja funckji powinna zawierać się powyżej miejsca jej wykorzystania.

### Tworzenie prostych funkcji

W Pythonie 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.

|  |
| --- |
| [module\_5\_lesson\_1\_simple\_function.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.

|  |
| --- |
| [module\_5\_lesson\_1\_simple\_return\_function.py] |

Funkcja w Pythonie może zwracać więcej niż jedną wartość. W takim przypadku zwraca krotkę, którą można rozpakować, aby uzyskać dostęp do poszczególnych elementów.

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

### Dokumentowanie funkcji za pomocą *docstring’ów*

Komentarze w kodzie są bardzo przydatne, gdy chcemy zrozumieć kolejność wykonywanych operacji. Służą jako notatki autora, co ułatwia powrót do pracy nad kodem nawet po dłuższej przerwie. Stanowią one formę dokumentacji, którą później łatwo jest wykorzystać. Bardziej rozbudowaną formą są tzw. *Docstringi* czyli łańcuchy znaków dokumentujące zawartość, najczęściej i w większości funkcji.

*Docstring* tworzy się wewnątrz potrójnego cudzysłowiu *”””* oznaczającego początek i koniec dokumentacji. Powinien składać się z trzech części:

* Ogólnego opisu co robi funkcja.
* Listę parametrów jakie funkcja przyjmuje.
* Co zwraca.

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

### Zadania

**Zadanie 5.1.1. Funkcja mnożąca**

Napisz funkcję która:

* Jako argumenty przyjmie dwie liczby i zwróci ich iloczyn
* Dodaj docstring z opisem działania

**Zadanie 5.1.2. Konwersja stopni Celsjusza na Fahrenheita**

Napisz funkcję która:

* Przelicza temperaturę podaną w stopniach Celsjusza na Fahrenheita i zwraca wynik (podpowiedź: *F = (C × 9/5) + 32*)
* Dodaj docstring z opisem działania

**Zadanie 5.1.3. Sprawdzanie parzystości**

Napisz funkcję, która:

* przyjmuje liczbę całkowitą i zwraca wartość *True*, jeśli liczba jest parzysta, lub *False* w przeciwnym przypadku
* Dodaj docstring opisujący działanie funkcji.

**Zadanie 5.1.4. \* Zliczanie znaków w zdaniu**

Napisz funkcję, która:

* Przyjmie zdanie jako ciąg znaków i zwróci liczbę słów w nim zawartych
* W ciele sprawdzaj czy argument jest typu *str.* Jeżeli nie, wypisz w konsoli informacje o nieprawidłowym typie podanego argumentu
* Dodaj docstring opisujący działanie funkcji

**Zadanie 5.1.5. \* Łączenie słowników**

Napisz funkcję, która:

* Łączy ze sobą dwa słowniki podane jako argumenty tej funkcji i zwraca w postaci jednego słownika
* Funkcja powinna łączyć słowniki, nawet gdy pary klucz-wartość nie są takie same.
* Dodaj docstring opisujący jak funkcja działa

## 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*

Składania pętli *for* składa się ze wspomnianego słowa kluczowego po którym następuje nazwa zmiennej, do którego przypisana będzie wartość aktualnie pobranego elementu z iterowanego elementu. Następnie występuje słowo kluczowe *in* a za nim iterowany obiekt. Po dwukropku i wcięciu występuje wykonywany blok kodu.

|  |
| --- |
| 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 i przechodzi do kolejnego elementu. Jeżeli pobrany element w aktualnej iteracji

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

Składnia pętli *while* składa się ze wspomnianego słowa kluczowego i warunku. Pętla ta wykonuje się do momentu dopóki warunek jest spełniony (tj. zwraca wartość *true*). Następnie po dwukropku i wcięciu następuje wykonywany blok kodu.

|  |
| --- |
| while warunek:  # kod, który będzie wykonywany, dopóki warunek jest spełniony |

Działanie pętli *while* polega na tym, że przed każdą iteracją sprawdzany jest warunek. Jeśli warunek jest prawdziwy, wykonywany jest blok kodu, a następnie warunek jest ponownie oceniany. Proces ten trwa do momentu, gdy warunek przestanie być spełniony, co powoduje zakończenie pętli. Aby uniknąć nieskończonej pętli, należy zadbać, aby wewnątrz bloku kodu zmieniana była wartość wpływająca na warunek lub używać instrukcji *break*, która przerywa działanie pętli. Wyjątkiem od tego stanowią algorytmy działania, które w założeniu mają działać cały czas, do momentu zastopowania działania programu (np. program działania robota).

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

### Zadania iteracyjne z zastosowaniem funkcji

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

|  |
| --- |
| [module\_5\_lesson\_2\_sum\_list\_elem.py] |

lub literowanie wyrazów.

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

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

|  |
| --- |
| [module\_5\_lesson\_2\_loop\_user\_input.py] |

Bardziej zaawansowanym, użyciem pętli będzie wykorzystanie funkcji wbudowanej *enumerate()*, która jako argument przyjmuje taki iterowalny obiekt, natomiast zwraca nie jeden a dwa elementy. Pierwszy z nich to index pobranego elementu, a drugi to sam element. Metoda ta jest bardzo przydatna przy działaniu na takich zbiorach jak słowniki.

|  |
| --- |
| [module\_5\_lesson\_2\_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\_5\_lesson\_2\_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 dla tworzenia interaktywnych terminali, gdzie w zależności od wprowadzonego wejścia można kontynuować działanie programu w nieskończonej pętli.

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

### Zadania

**Zadanie 5.2.1. 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 5.2.2. Odwracanie kolejności znaków**

Napisz program, który:

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

**Zadanie 5.2.3. Wybieranie liczb**

Napisz program, w którym:

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

**Zadanie 5.2.4. \* 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 5.2.5. \* 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\_5\_lesson\_3\_fibonacci.py] |

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

|  |
| --- |
| [module\_5\_lesson\_3\_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\_5\_lesson\_3\_global\_local\_var.py] |

### Zadania

**Zadanie 5.3.1. Rekurencyjna silnia**

Napisz rekurencyjną funkcję, która:

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

**Zadanie 5.3.2. Rekurencyjne sumowanie**

Napisz funkcję, która:

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

**Zadanie 5.3.3. 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 5.3.4. \* Rekurencyjne sumowanie cyfr liczby**

Napisz funkcję rekurencyjną, która:

* Oblicza sumę cyfr podanej liczby.

**Zadanie 5.3.5. \* Rekurencyjne obliczenie największego wspólnego dzielnika**

Napisz funkcję rekurencyjną, która:

* Oblicza największy wspólny dzielnik dwóch liczb.

## Podsumowanie

W tym rozdziale dowiedzieliśmy się:

* Jak tworzy się funkcje w języku Python i jak za ich pomocą można zwracać obliczone wartości .
* Jak tworzyć *docstring’i* i za ich pomocą dokumentować co robi dana funkcja.
* W jezyku Python występują dwa typy pętli: *for* i *while.*
* Pętla *for* jest stworzona do iterowania na zdolnych do tego obiektach a pętla *while* jest wykonywana dopóki spełniony jest jej warunek działania.
* Rekurencja czyli samo odwołanie się do funkcji jest użyteczne przy rozwiązywaniu niektórych problemów algorytmicznych
* Zmienne lokalne mają ograniczony czas życia, kiedy zmienne globalne są dostępne przez cały czas działania programu
* Wewnątrz funkcji można zmieniać wartości zmiennej globalnej za pomocą słowa kluczowego *global*

## Zadania

**Zadanie 5.5.1. Odliczanie w dół**

Stwórz funkcję count\_down(n), która:

* Przy użyciu pętli while wypisuje liczby od n do 1.
* Dodaj docstring opisujący działanie funkcji.

**Zadanie 5.5.2 \* Zliczanie samogłosek.**

Utwórz funkcję count\_vowels(word), która:

* Iteruje po znakach słowa i zwraca liczbę samogłosek.
* Uzupełnij funkcję o docstring.

**Zadanie 5.5.3. Wartość bezwzględna**

Utwórz funkcję absolute\_value(n), która:

* Zwraca wartość bezwzględną liczby *n*.
* Użyj warunków i dołącz docstring.

**Zadanie 5.5.4. Łączenie list**

Napisz funkcję concatenate\_lists(list1, list2), która:

* Łączy dwie listy w jedną, wykorzystując pętlę for

**Zadanie 5.5.5. Filtrowanie liczb parzystych**

Stwórz funkcję filter\_even\_numbers(lst), która:

* Zwraca nową listę zawierającą tylko parzyste liczby z podanej listy

**Zadanie 5.5.6. Szukanie minimum**

Utwórz funkcję find\_min(lst), która:

* Znajduje najmniejszy element w liście przy użyciu pętli for.
* Uzupełnij funkcję o docstring.

**Zadanie 5.5.7. \* Sprawdzanie liczby pierwszej**

Utwórz funkcję is\_prime(n), która:

* Przy użyciu pętli for sprawdza, czy liczba n jest pierwsza.
* Funkcja powinna zwracać True, jeśli liczba jest pierwsza, i False w przeciwnym razie.
* Dołącz docstring.

**Zadanie 5.5.8. \* Sprawdzanie palindromu**

Napisz funkcję is\_palindrome\_iterative(s), która:

* Za pomocą pętli sprawdza, czy dany łańcuch znaków jest palindromem.
* Dołącz docstring wyjaśniający działanie funkcji.

**Zadanie 5.5.9. Wyliczanie wagi**

Utwórz funkcję weighted\_average(values, weights), która:

* Oblicza średnią ważoną na podstawie dwóch list – jednej z wartościami i drugiej z odpowiadającymi im wagami.
* Upewnij się, że listy mają taką samą długość.

**Zadanie 5.5.10. Zwracanie liczb podzielnych przez 3**

Napisz funkcję divisible\_by\_three(lst), która:

* Przyjmuje listę liczb i zwraca nową listę zawierającą tylko liczby podzielne przez 3.

**Zadanie 5.5.11. \* Generowanie tabliczki mnożenia**

Stwórz funkcję print\_multiplication\_table(n), która:

* Generuje tabliczkę mnożenia dla liczb od 1 do n przy użyciu zagnieżdżonych pętli for i wyświetla wynik w formie macierzy

**Zadanie 5.5.12. \* Łączenie słów**

Napisz funkcję join\_words(words), która:

* Przy użyciu pętli for łączy elementy listy łańcuchów znaków w jeden tekst, oddzielając je spacjami.

**Zadanie 5.5.13. Podwajanie liter**

Utwórz funkcję double\_letters(text), która:

* Iteruje po podanym łańcuchu znaków i zwraca nowy tekst, w którym każda litera występuje podwójnie.

**Zadanie 5.5.14. \* Wyznaczanie elementu ciągu**

Napisz funkcję recursive\_sequence(n), która:

* Definiuje ciąg rekurencyjny według reguły: a(0)=1, a(n)=a(n-1)+2.
* Funkcja powinna zwracać n-ty wyraz ciągu.

**Zadanie 5.5.15. \* Generowanie ciągu geometrycznego**

Napisz funkcję geometric\_sequence(a, r, n), która:

* Generuje listę n wyrazów ciągu geometrycznego zaczynając od liczby a i o ilorazie r przy użyciu pętli for.

**Zadanie 5.5.16. Kwadrat elementów listy**

Stwórz funkcję square\_list(numbers), która:

* Przyjmuje listę liczb i zwraca nową listę, w której każdy element jest kwadratem odpowiadającego elementu wejściowej listy.

**Zadanie 5.5.17. Zliczanie liter**

Utwórz funkcję count\_letters(word), która:

* Zlicza liczbę liter w podanym słowie (pomijając spacje).
* Użyj pętli for

**Zadanie 5.5.18. Wyznaczanie średniej**

Napisz funkcję calculate\_average(lst), która:

* Oblicza średnią wartość elementów listy liczb przy użyciu pętli for

**Zadanie 5.5.19. \* Pobieranie od użytkownika liczb w pętli**

Utwórz funkcję input\_numbers(), która:

* W pętli while pobiera liczby od użytkownika, dopóki nie wpisze on słowa "exit".
* Funkcja powinna zwracać listę wprowadzonych liczb.

**Zadanie 5.5.20. \* Rekurencyjna suma cyfr aż do uzyskania liczby jednocyfrowej**

Napisz funkcję recursive\_digit\_sum(n), która:

* Rekurencyjnie sumuje cyfry liczby, aż wynik będzie jednocyfrowy, i zwraca tę wartość.
* Upewnij się, że funkcja ma poprawnie zdefiniowany warunek zakończenia oraz dołącz docstring.

## 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] |

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

**Zadanie 5.1.4.**

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

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

**Zadanie 5.1.5.**

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

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

**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\_3\_5\_3\_1.py] |

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

**Zadanie 5.3.2.**

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

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

**Zadanie 5.3.3.**

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

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

**Zadanie 5.3.4.**

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

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

**Zadanie 5.3.5.**

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

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

**Zadanie 5.5.1.**

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

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

**Zadanie 5.5.2.**

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

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

**Zadanie 5.5.3.**

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

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

**Zadanie 5.5.4.**

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

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

**Zadanie 5.5.5.**

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

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

**Zadanie 5.5.6.**

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

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

**Zadanie 5.5.7.**

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

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

**Zadanie 5.5.8.**

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

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

**Zadanie 5.5.9.**

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

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

**Zadanie 5.5.10.**

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

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

**Zadanie 5.5.11.**

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

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

**Zadanie 5.5.12.**

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

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

**Zadanie 5.5.13.**

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

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

**Zadanie 5.5.14.**

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

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

**Zadanie 5.5.15.**

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

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

**Zadanie 5.5.16.**

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

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

**Zadanie 5.5.17.**

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

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

**Zadanie 5.5.18.**

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

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

**Zadanie 5.5.19.**

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

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

**Zadanie 5.5.20.**

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

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

# Operacje na systemach plików

Wprowadzenie do obsługi systemu plików w języku Python obejmuje zarówno tworzenie, jak i przeszukiwanie katalogów. Pliki można generować przy użyciu funkcji *open()* z trybem *w* lub *x*, zapewniając kontrolę nad ich nadpisywaniem. Struktura katalogów jest zarządzana przez moduł *os*, który umożliwia wykorzystanie metod *mkdir()* oraz *makedirs()*. Przeglądanie zawartości folderów i wyszukiwanie plików o określonych rozszerzeniach odbywa się za pomocą modułów os oraz *pathlib*, które zapewniają wygodne operacje na ścieżkach.

## Tworzenie plików i katalogów

System plików w Pythonie pozwala na dynamiczne zarządzanie danymi poprzez tworzenie i edytowanie plików oraz katalogów. Tworzenie nowych plików odbywa się za pomocą funkcji *open()*, w której tryb *w (write)* pozwala na zapis, a *x (exclusive creation)* zapobiega nadpisaniu już istniejących danych. Dla organizacji struktury katalogów Python udostępnia moduł *os*, który oferuje metody *mkdir()* do tworzenia pojedynczego folderu oraz *makedirs()* umożliwiającą tworzenie całej hierarchii katalogów jednocześnie. Dzięki tym narzędziom programista może w pełni kontrolować strukturę plików i katalogów, dostosowując ją do potrzeb aplikacji.

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

### Zadania

**Zadanie 6.1.1. Tworzenie pliku i zapis tekstu**

Napisz program, który utworzy plik *dane.txt*, a następnie zapisze w nim tekst: "To jest przykładowy plik utworzony w Pythonie.".

**Zadanie 6.1.2. Tworzenie katalogu**

Napisz skrypt, który sprawdzi, czy istnieje katalog o nazwie dokumenty. Jeśli nie istnieje, utworzy go za pomocą *os.mkdir()*.

**Zadanie 6.1.3. Tworzenie zagnieżdżonej struktury katalogów**

Rozszerz poprzednie zadanie, aby zamiast jednego katalogu skrypt tworzył całą strukturę katalogów dokumenty/kurs/2025, wykorzystując os.makedirs().

**Zadanie 6.1.4. Tworzenie, zapis i odczyt pliku**

Napisz program, który utworzy plik log.txt, zapisze w nim trzy dowolne linie tekstu, a następnie odczyta i wyświetli jego zawartość.

**Zadanie 6.1.5. \* Wyszukiwanie plików o określonym rozszerzeniu**

Napisz skrypt, który przeszuka bieżący katalog i wyświetli listę wszystkich plików o rozszerzeniu .txt. Wykorzystaj do tego moduł os lub pathlib.

## Przeszukiwanie zawartości katalogu

Przeglądanie zawartości katalogu w można zrealizować za pomocą modułu *os*, który umożliwia pobieranie listy plików i folderów. Moduł *pathlib* dostarcza bardziej obiektowego podejścia do pracy ze ścieżkami, umożliwiając filtrowanie plików według rozszerzeń. Dzięki metodom takim jak *os.listdir()* czy *Path.glob()* można efektywnie wyszukiwać określone pliki w systemie plików.

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

### Zadania

**Zadanie 6.2.1. Wyświetlanie zawartości katalogu**

Napisz program, który wyświetli listę wszystkich plików i folderów w katalogu Pulpit przy użyciu funkcji *os.listdir().*

**Zadanie 6.2.2. Wyszukiwanie plików o określonym rozszerzeniu**

Napisz skrypt, który znajdzie wszystkie pliki z rozszerzeniem *.txt* w bieżącym katalogu, korzystając z modułu *pathlib*.

**Zadanie 6.2.3. Przeszukiwanie podkatalogów**

Rozszerz poprzednie zadanie, aby skrypt wyszukiwał pliki *.txt* nie tylko w bieżącym katalogu, ale także we wszystkich jego podkatalogach. Użyj *Path.rglob()*.

**Zadanie 6.2.4. \* Filtracja plików według rozmiaru**

Napisz program, który znajdzie i wyświetli listę plików większych niż 100 KB w bieżącym katalogu, wykorzystując *os.path.getsize()*.

**Zadanie 6.2.5. \* Zliczanie liczby plików w katalogu**

Napisz skrypt, który policzy, ile plików znajduje się w bieżącym katalogu oraz ile z nich ma rozszerzenie *.py*. Wykorzystaj *os* lub *pathlib*.

## Praca z plikami tekstowymi

Operacje na plikach tekstowych są nieodłącznym elementem zarządzania danymi w Pythonie. Do odczytu zawartości plików służy funkcja open() z trybem r (read), który umożliwia pobranie zawartości pliku linia po linii lub w całości. Zapisywanie danych do plików można realizować w trybie w (write) lub a (append), gdzie w nadpisuje plik, z kolei *a* dopisuje nowe dane na końcu. Podczas pracy z plikami warto stosować obsługę wyjątków (try-except), aby unikać błędów związanych np. z nieistniejącym plikiem lub brakiem uprawnień do zapisu.

|  |  |
| --- | --- |
| 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. |

|  |
| --- |
| [module\_6\_lesson\_2\_text\_files.py] |

### Zadania

**Zadanie 6.3.1. Odczyt i wyświetlenie zawartości pliku**

Napisz program, który otworzy plik *log.txt* w trybie r, a następnie wyświetli jego zawartość linia po linii.

**Zadanie 6.3.2. 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 6.3.3. 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 6.3.4. \* 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 6.3.5. \* 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.*

## Operacje na danych czasowych

Wbudowany moduł *datetime* pozwala na operacje związane z datą i czasem, takie jak tworzenie, formatowanie oraz wykonywanie obliczeń na datach. Jest często wykorzystywany w aplikacjach do rejestrowania zdarzeń, logowania aktywności użytkowników czy organizacji plików według czasu.

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

### Zadania

**Zadanie 6.4.1. Pobranie i formatowanie aktualnej daty i czasu**

Napisz program, który pobierze aktualną datę i godzinę, a następnie wyświetli ją w formacie DD-MM-YYYY HH:MM:SS.

**Zadanie 6.4.2. Obliczanie różnicy między dwiema datami**

Napisz program, który poprosi użytkownika o dwie daty w formacie YYYY-MM-DD, a następnie obliczy różnicę między nimi w dniach.

**Zadanie 6.4.3. Dodawanie dni do daty**

Napisz program, który pobierze aktualną datę i doda do niej 7 dni.

**Zadanie 6.4.4. Tworzenie pliku z datą w nazwie**

Napisz skrypt, który utworzy plik o nazwie w formacie backup\_YYYY-MM-DD\_HH-MM-SS.txt i zapisze w nim aktualną datę i godzinę.

**Zadanie 6.4.5. \* Wyszukiwanie plików utworzonych w określonym przedziale czasowym**

Napisz program, który znajdzie i wyświetli listę plików w bieżącym katalogu, które zostały utworzone lub zmodyfikowane w ciągu ostatnich 24 godzin. Wykorzystaj moduł *os* i *datetime*.

## Podsumowanie

Moduł os i *pathlib* umożliwiają tworzenie, modyfikację oraz przeszukiwanie plików i katalogów, oferując funkcje takie jak *mkdir()*, *makedirs()*, *listdir()* i *glob()*. Praca z plikami tekstowymi odbywa się za pomocą funkcji open(), która obsługuje tryby odczytu, zapisu i dopisywania (*r*, *w*, *a*, *r+*, *w+*, *a+*). Moduł *shutil* pozwala na kopiowanie, przenoszenie i kompresowanie plików, a dodatkowe operacje, takie jak sortowanie plików według daty modyfikacji, można realizować przy użyciu *os.path.getmtime()*. Dodatkowo, moduł *datetime* umożliwia formatowanie dat, obliczanie różnic między datami oraz automatyczne tworzenie plików i katalogów z uwzględnieniem bieżącego czasu.

## Zadania

**Zadanie 6.6.1. Sprawdzanie istnienia pliku i odczyt zawartości**

* Napisz program, który sprawdzi, czy istnieje plik dane.txt. Jeśli plik istnieje, odczyta i wyświetli jego zawartość. Jeśli nie istnieje, poinformuje użytkownika o jego braku.

**Zadanie 6.6.2. Kopiowanie pliku do innej lokalizacji**

* Napisz program, który skopiuje plik *dane.txt* do katalogu *dokumenty*. Jeśli katalog nie istnieje, utwórz go przed kopiowaniem. Wykorzystaj moduł *shutil*.

**Zadanie 6.6.3. Usuwanie pliku**

* Napisz skrypt, który usunie plik dane2.txt, jeśli istnieje. Jeśli plik nie istnieje, wyświetli odpowiedni komunikat.

**Zadanie 6.6.4. Zmiana nazwy pliku**

* Napisz skrypt, który zmieni nazwę pliku dane.txt na archiwum.txt. Jeśli plik dane.txt nie istnieje, wyświetl odpowiedni komunikat.

**Zadanie 6.6.5. Przeszukiwanie katalogu pod kątem plików o określonym rozszerzeniu**

* Napisz program, który wyszuka wszystkie pliki *.log* w katalogu *dokumenty* i wyświetli ich listę. Jeśli katalog nie istnieje, poinformuje użytkownika o błędzie.

**Zadanie 6.6.6. Pobieranie informacji o plikach**

* Napisz skrypt, który dla wszystkich plików w katalogu *dokumenty* wyświetli ich nazwy, rozmiary w kilobajtach oraz datę ostatniej modyfikacji. Jeśli katalog nie istnieje, program powinien poinformować użytkownika..

**Zadanie 6.6.7. Tworzenie wielu plików jednocześnie**

* Napisz skrypt, który utworzy 5 plików (*plik1.txt*, *plik2.txt*, ..., plik5.txt) i zapisze w nich tekst: "To jest plik numer X", gdzie X to numer pliku.

**Zadanie 6.6.8. Tworzenie zagnieżdżonej struktury katalogów**

* Napisz skrypt, który utworzy strukturę katalogów *Kurs/Python/Projekt*.

**Zadanie 6.6.9. Przenoszenie plików między katalogami**

* Napisz program, który przeniesie plik dane.txt do katalogu *Kurs/Python/Projekt*, tworząc katalog, jeśli nie istnieje.

**Zadanie 6.6.10. Odczyt i zapis pliku linia po linii**

* Napisz program, który otworzy plik *dane.txt*, odczyta go linia po linii i zapisze jego zawartość do nowego pliku *kopia\_dane.txt*.

**Zadanie 6.6.11. Sprawdzenie, czy rok jest przestępny i zapis do pliku**

* Napisz program, który sprawdzi czy aktualny rok jest przestępny. (Rok jest przestępny, jeśli jest podzielny przez 4, ale nie przez 100, chyba że jest podzielny przez 400).

**Zadanie 6.6.12. Zliczanie liczby plików w katalogu**

* Napisz skrypt, który policzy, ile plików znajduje się w katalogu *dokumenty* oraz ile z nich ma rozszerzenie *.txt*.

**Zadanie 6.6.13. Usuwanie pustych katalogów**

* Napisz skrypt, który znajdzie i usunie wszystkie puste katalogi w folderze dokumenty/. Jeśli katalogi nie istnieją lub są niepuste, wyświetl odpowiedni komunikat.

**Zadanie 6.6.14. Tworzenie pliku tymczasowego**

* Napisz program, który utworzy plik tymczasowy, zapisze w nim tekst "To jest plik tymczasowy" i automatycznie go usunie po zamknięciu.

**Zadanie 6.6.15. Sprawdzanie uprawnień do pliku**

* Napisz skrypt, który sprawdzi, czy plik *dane.txt* ma uprawnienia do odczytu i zapisu, a następnie wyświetli odpowiedni komunikat.

**Zadanie 6.6.16. Sprawdzanie, czy podana ścieżka to plik czy katalog**

* Napisz skrypt, który poprosi użytkownika o podanie ścieżki i sprawdzi, czy jest to plik czy katalog.

**Zadanie 6.6.17. Sortowanie plików w katalogu według daty modyfikacji**

* Napisz program, który wyświetli listę plików w katalogu *dokumenty* posortowaną według daty ostatniej modyfikacji.

**Zadanie 6.6.18. Kompresowanie plików do archiwum ZIP**

* Napisz skrypt, który skompresuje wszystkie pliki *.txt* z katalogu *dokumenty* do archiwum *backup.zip*.

**Zadanie 6.6.19. Dekompresowanie archiwum ZIP**

* Napisz program, który rozpakowuje pliki z backup.zip do katalogu *odzyskane*.

**Zadanie 6.6.20. Zmiana rozszerzenia wszystkich plików w katalogu**

* Napisz skrypt, który zmieni rozszerzenie wszystkich plików *.txt* w katalogu *dokumenty* na *.bak*.

## 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] |

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

**Zadanie 6.2.1.**

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

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

**Zadanie 6.2.2.**

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

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

**Zadanie 6.2.3.**

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

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

**Zadanie 6.2.4.**

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

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

**Zadanie 6.2.5.**

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

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

**Zadanie 6.3.1.**

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

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

**Zadanie 6.3.2.**

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

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

**Zadanie 6.3.3.**

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

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

**Zadanie 6.3.4.**

|  |
| --- |
| [module\_6\_lesson\_2\_6\_3\_4.py] |

[module\_6\_lesson\_2\_6\_3\_4.png]

**Zadanie 6.3.5.**

|  |
| --- |
| [module\_6\_lesson\_2\_6\_3\_5.py] |

[module\_6\_lesson\_2\_6\_3\_5.png]

**Zadanie 6.4.1.**

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

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

**Zadanie 6.4.2.**

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

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

**Zadanie 6.4.3.**

|  |
| --- |
| [module\_6\_lesson\_2\_6\_4\_3.py] |

[module\_6\_lesson\_2\_6\_4\_3.png]

**Zadanie 6.4.4.**

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

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

**Zadanie 6.4.5.**

|  |
| --- |
| [module\_6\_lesson\_2\_6\_4\_5.py] |

[module\_6\_lesson\_2\_6\_4\_5.png]

**Zadanie 6.6.1.**

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

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

**Zadanie 6.6.2.**

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

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

**Zadanie 6.6.3.**

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

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

**Zadanie 6.6.4.**

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

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

**Zadanie 6.6.5.**

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

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

**Zadanie 6.6.6.**

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

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

**Zadanie 6.6.7.**

|  |
| --- |
| [module\_6\_summary\_6\_6\_7.py] |

[module\_6\_summary\_6\_6\_7.png]

**Zadanie 6.6.8.**

|  |
| --- |
| [module\_6\_summary\_6\_6\_8.py] |

[module\_6\_summary\_6\_6\_8.png]

**Zadanie 6.6.9.**

|  |
| --- |
| [module\_6\_summary\_6\_6\_9.py] |

[module\_6\_summary\_6\_6\_9.png]

**Zadanie 6.6.10.**

|  |
| --- |
| [module\_6\_summary\_6\_6\_10.py] |

[module\_6\_summary\_6\_6\_10.png]

**Zadanie 6.6.11.**

|  |
| --- |
| [module\_6\_summary\_6\_6\_11.py] |

[module\_6\_summary\_6\_6\_11.png]

**Zadanie 6.6.12.**

|  |
| --- |
| [module\_6\_summary\_6\_6\_12.py] |

[module\_6\_summary\_6\_6\_12.png]

**Zadanie 6.6.13.**

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

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

**Zadanie 6.6.14.**

|  |
| --- |
| [module\_6\_summary\_6\_6\_14.py] |

[module\_6\_summary\_6\_6\_14.png]

**Zadanie 6.6.15.**

|  |
| --- |
| [module\_6\_summary\_6\_6\_15.py] |

[module\_6\_summary\_6\_6\_15.png]

**Zadanie 6.6.16.**

|  |
| --- |
| [module\_6\_summary\_6\_6\_16.py] |

[module\_6\_summary\_6\_6\_16.png]

**Zadanie 6.6.17.**

|  |
| --- |
| [module\_6\_summary\_6\_6\_17.py] |

[module\_6\_summary\_6\_6\_17.png]

**Zadanie 6.6.18.**

|  |
| --- |
| [module\_6\_summary\_6\_6\_18.py] |

[module\_6\_summary\_6\_6\_18.png]

**Zadanie 6.6.19.**

|  |
| --- |
| [module\_6\_summary\_6\_6\_19.py] |

[module\_6\_summary\_6\_6\_19.png]

**Zadanie 6.6.20.**

|  |
| --- |
| [module\_6\_summary\_6\_6\_20.py] |

[module\_6\_summary\_6\_6\_20.png]

## 

Wytyczne

1. ..., **Wykonawca przygotuje do nich materiały szkoleniowe (e-prezentacje i e-skrypty) na szkolenie „Nauka programowania w językach Scratch i Python” dla trenerów oraz nauczycielek/nauczycieli będących uczestnikami szkoleń** oraz końcowe testy, o których mowa pkt 4.
2. **obejmować m.in. e-prezentacje dla trenerów prowadzących szkolenia oraz e-skrypty-materiały tekstowe powiązane z klipami wideo zgodne z treściami edukacyjnymi dla nauczycieli/nauczycielek uczestniczących w szkoleniu**
3. **Tematy *rozkładu materiału nauczania* powinny być pogrupowane w rozdziały, gdzie rozdział odpowiada jednemu tematowi *ramowego programu szkolenia*, który będzie tytułem rozdziału.** Analogicznie, tekstowe *materiały szkoleniowe* dotyczące tematów objętych rozdziałami *rozkładu materiału nauczania* powinny być pogrupowane w rozdziały o tych samych tytułach.
   1. **Opis każdego tematu lub zagadnienia powinien obejmować przynajmniej jeden przykład dotyczący tego tematu lub zagadnienia. Przykłady muszą zostać przedstawione w formie tekstowej oraz graficznej.** Przykłady odnoszące się do programowania powinny zawierać: treść zadania programistycznego; opis danych wejściowych i wynikowych; opis algorytmu; skrypt w danym języku programowania realizujący algorytm oraz obraz graficzny ekranu z wynikami działania skryptu.
   2. **Opis każdego tematu musi kończyć się co najmniej 5 *zadaniami praktycznymi* dla uczestnika/uczestniczki**.

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)