Nauka programowania w językach Scratch i Python

Python – poziom podstawowy

Materiał dla nauczycieli

Spis treści

[1. Podstawy języka Python 4](#_Toc190304700)

[1.1. Język Python, jego rola i podstawowe narzędzia 4](#_Toc190304701)

[1.1.1. Wprowadzenie 4](#_Toc190304702)

[1.1.1.1. Struktura kursu i sposób nauki 4](#_Toc190304703)

[1.1.1.2. Zastosowanie języka 4](#_Toc190304704)

[1.1.1.3. Wady i zalety języka Python 5](#_Toc190304705)

[1.1.1.4. Wykorzystanie języka Python – interaktywnie czy przez skrypt? 6](#_Toc190304706)

[1.1.2. Algorytmika – czym jest? 6](#_Toc190304707)

[1.1.3. Podstawowe narzędzia kursu 8](#_Toc190304708)

[1.1.3.1. Systemy operacyjne (Windows oraz Ubuntu) 8](#_Toc190304709)

[1.1.3.2. Środowiska i edytory (Thonny, Visual Studio Code) 8](#_Toc190304710)

[1.1.3.3. Uruchamianie kodu języka Python w terminalu 10](#_Toc190304711)

[1.1.4. Zadania praktyczne 11](#_Toc190304712)

[1.2. Instalacja środowiska Python w różnych systemach operacyjnych 12](#_Toc190304713)

[1.2.1. Instalacja środowiska Python na systemie Linux 12](#_Toc190304714)

[1.2.2. Instalacja środowiska Python na systemie Windows 16](#_Toc190304715)

[1.2.3. Weryfikacja wersji interpretera języka Python 19](#_Toc190304716)

[1.2.4. Tworzenie wirtualnych środowisk 20](#_Toc190304717)

[1.2.5. Zarządzanie wersjami 21](#_Toc190304718)

[1.2.6. Zadania 22](#_Toc190304719)

[1.3. Zasady tworzenia i struktury skryptów w języku Python 23](#_Toc190304720)

[1.3.1. Struktura plików .py 23](#_Toc190304721)

[1.3.2. Pierwsze skrypty w edytorze tekstowym 24](#_Toc190304722)

[1.3.3. Uruchamianie skryptów Python w konsoli systemowej 24](#_Toc190304723)

[1.3.4. Wybór trybu pracy: interaktywny czy skryptowy? 25](#_Toc190304724)

[1.3.5. Zadania podsumowujące temat 27](#_Toc190304725)

[1.4. Uruchamianie kodu w konsoli za pomocą interpretera 29](#_Toc190304726)

[1.4.1. Praca z interaktywną konsolą Python 29](#_Toc190304727)

[1.4.2. Wprowadzanie pojedynczych komend i natychmiastowe obserwowanie wyników 30](#_Toc190304728)

[1.4.3. Testowanie fragmentów kodu w trybie interaktywnym 30](#_Toc190304729)

[1.4.4. Zadania podsumowujące temat 32](#_Toc190304730)

[1.5. Prowadzenie operacji matematycznych i manipulacji tekstowych 33](#_Toc190304731)

[1.5.1. Podstawowe operacje arytmetyczne w języku Python 33](#_Toc190304732)

[1.5.2. Manipulacja napisami 34](#_Toc190304733)

[1.5.3. Wykorzystanie wbudowanych funkcji: len() i type() 36](#_Toc190304734)

[1.5.4. Zadania podsumowujące temat 36](#_Toc190304735)

[1.6. Poznanie typów zmiennych w Pythonie 38](#_Toc190304736)

[1.6.1. Typy liczbowe 39](#_Toc190304737)

[1.6.2. Typy sekwencyjne 41](#_Toc190304738)

[1.6.3. Zbiory 43](#_Toc190304739)

[1.6.4. Typ mapujący 44](#_Toc190304740)

[1.6.5. Typ braku wartości 45](#_Toc190304741)

[1.6.6. Zadania 45](#_Toc190304742)

[1.7. Używanie metod wejścia i wyjścia 47](#_Toc190304743)

[1.7.1. Funkcja input() 47](#_Toc190304744)

[1.7.2. Funkcja print() 47](#_Toc190304745)

[1.7.3. Formatowanie stringów 48](#_Toc190304746)

[1.7.4. Zadania 49](#_Toc190304747)

[1.8. Korzystanie z bibliotek i menadżera PIP 50](#_Toc190304748)

[1.8.1. Instalacja i wykorzystanie bibliotek z pomocą narzędzia PIP 50](#_Toc190304749)

[1.8.2. Instalacja i korzystanie z bibliotek na przykładzie turtle 51](#_Toc190304750)

[1.8.3. Zadania 53](#_Toc190304751)

[1.9. Podsumowanie modułu 54](#_Toc190304752)

[1.10. Zadania podsumowujące moduł 55](#_Toc190304753)

[2. Rozpoczynamy kodowanie w języku Python 57](#_Toc190304754)

[2.1. Przygotowanie środowiska IDE 57](#_Toc190304755)

[2.1.1. Instalacja Thony i Visual Studio Code w systemie Linux 57](#_Toc190304756)

[2.1.2. Instalacja Thony i Visual Studio Code w systemie Windows 57](#_Toc190304757)

# 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 **8 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.

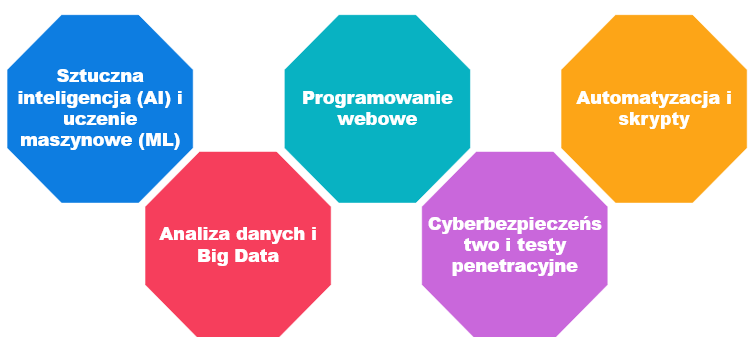
Sposób nauki opiera się na realizacji przykładów oraz samodzielnym wykonywaniu zadań utrwalających, co pozwala na aktywne przyswajanie materiału. Dla osób bardziej zaawansowanych przygotowano dodatkowe zadania (oznaczone \*) o wyższym stopniu trudności, które umożliwiają pogłębienie wiedzy i rozwój umiejętności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

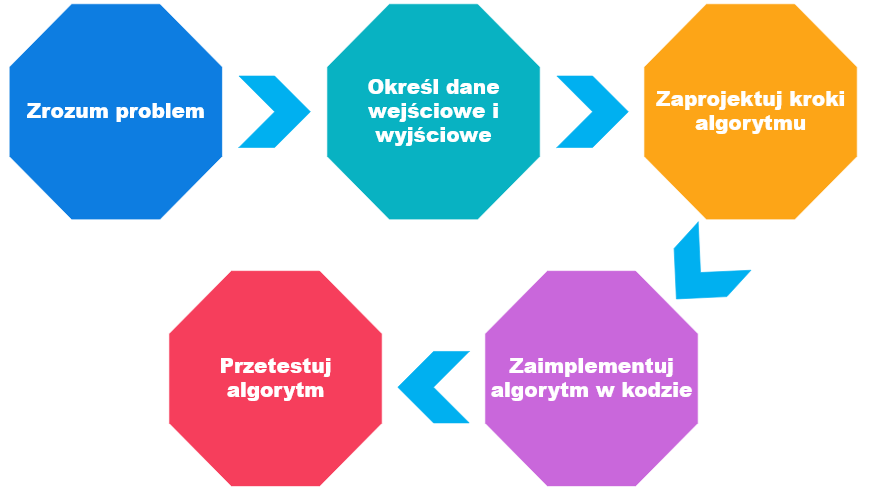
Algorytm to uporządkowany zbiór kroków lub instrukcji prowadzących do rozwiązania określonego problemu. Może być zapisany w postaci schematu blokowego, opisu słownego lub kodu komputerowego i powinien być jednoznaczny oraz efektywny. W języku Python algorytmy można implementować przy użyciu struktur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

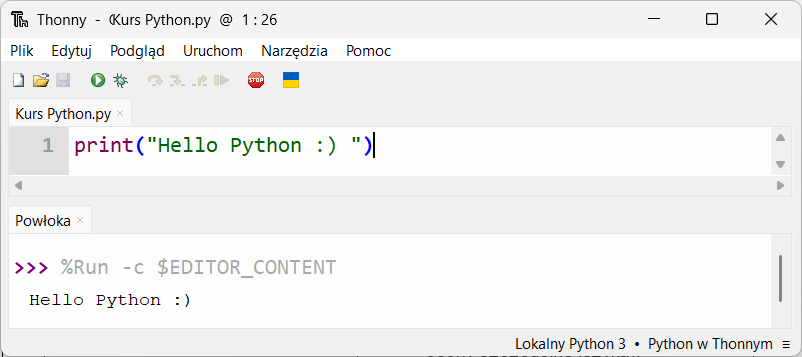
W trakcie kursu uczestnicy będą korzystać z różnych narzędzi wspomagających naukę i pracę z językiem Python. Obejmują one zarówno systemy operacyjne, na których działa Python, jak i środowiska programistyczne oraz edytory, które ułatwiają pisanie i uruchamianie kodu. W szczególności uczestnicy zapoznają 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.

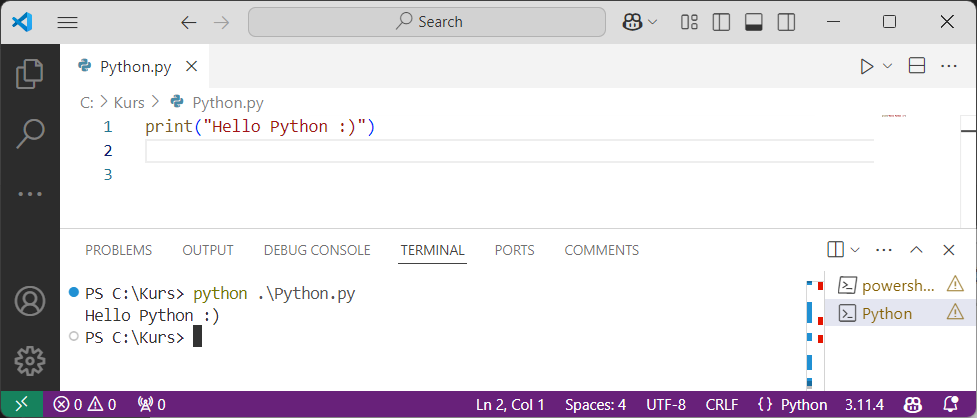
#### Środowiska i edytory (Thonny, Visual Studio Code)

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.

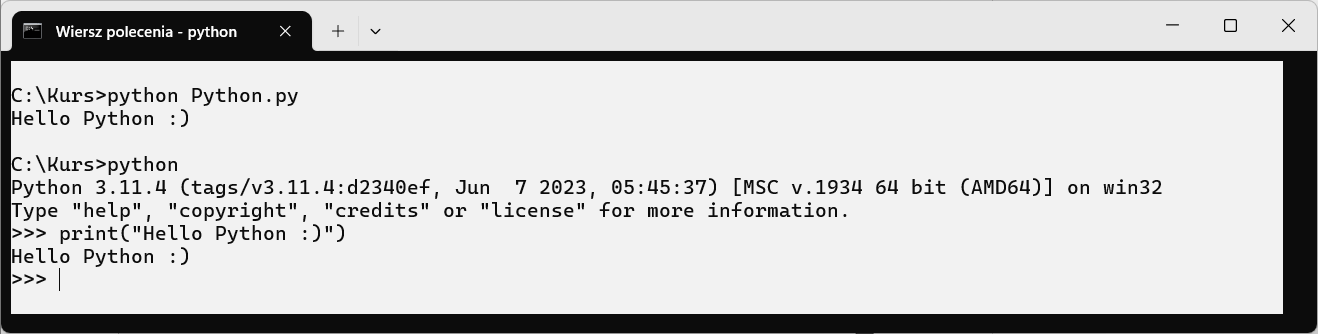


Rysunek 1.4. Visual Studio Code (VS Code) – lekki edytor z obsługą rozszerzeń

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 praktyczne

**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.

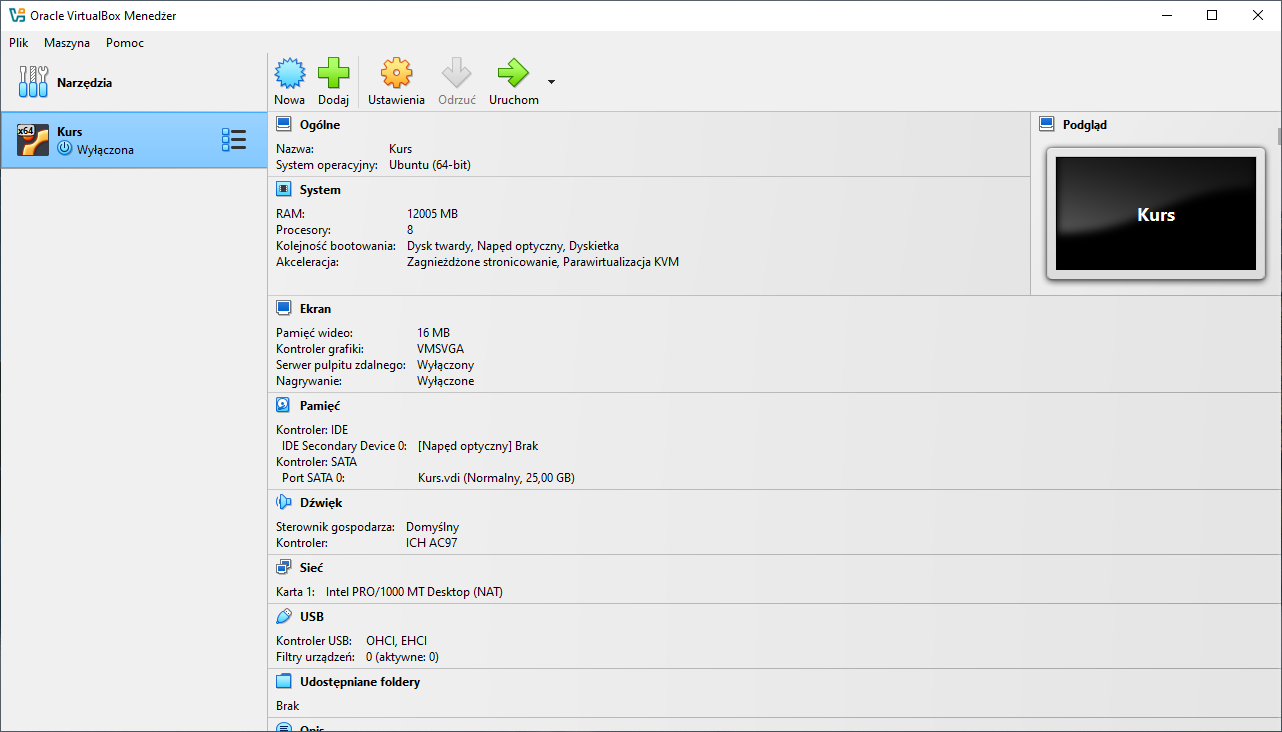
Aby skorzystać z systemu, należy uruchomić aplikację z poziomu pulpitu lub wpisać jej nazwę w pasku wyszukiwania systemu (Rysunek 1.6, Rysunek 1.1).

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

Rysunek 1.6. Ikona emulatora systemu Oracle VirtualBox

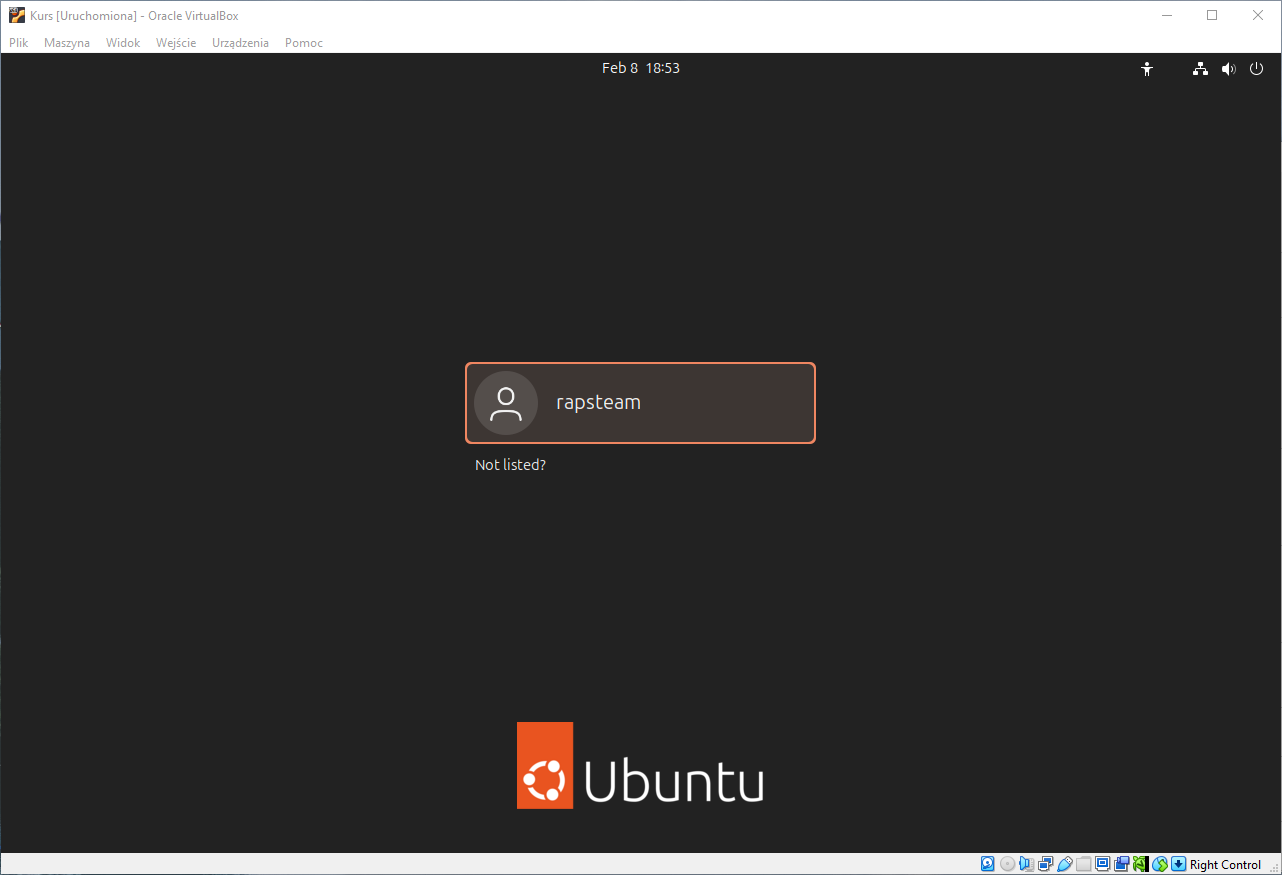
Po uruchomieniu, pojawi się okno z widokiem dostępnych i skonfigurowanych maszyn wirtualnych, gotowych do uruchomienia. Aby wystartować jedną z nich, należy wybrać pozycję z listy po prawej stronie widoku o nazwie *Kurs* i nacisnąć klawisz *Uruchom* widoczny w górnej części okna.

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.



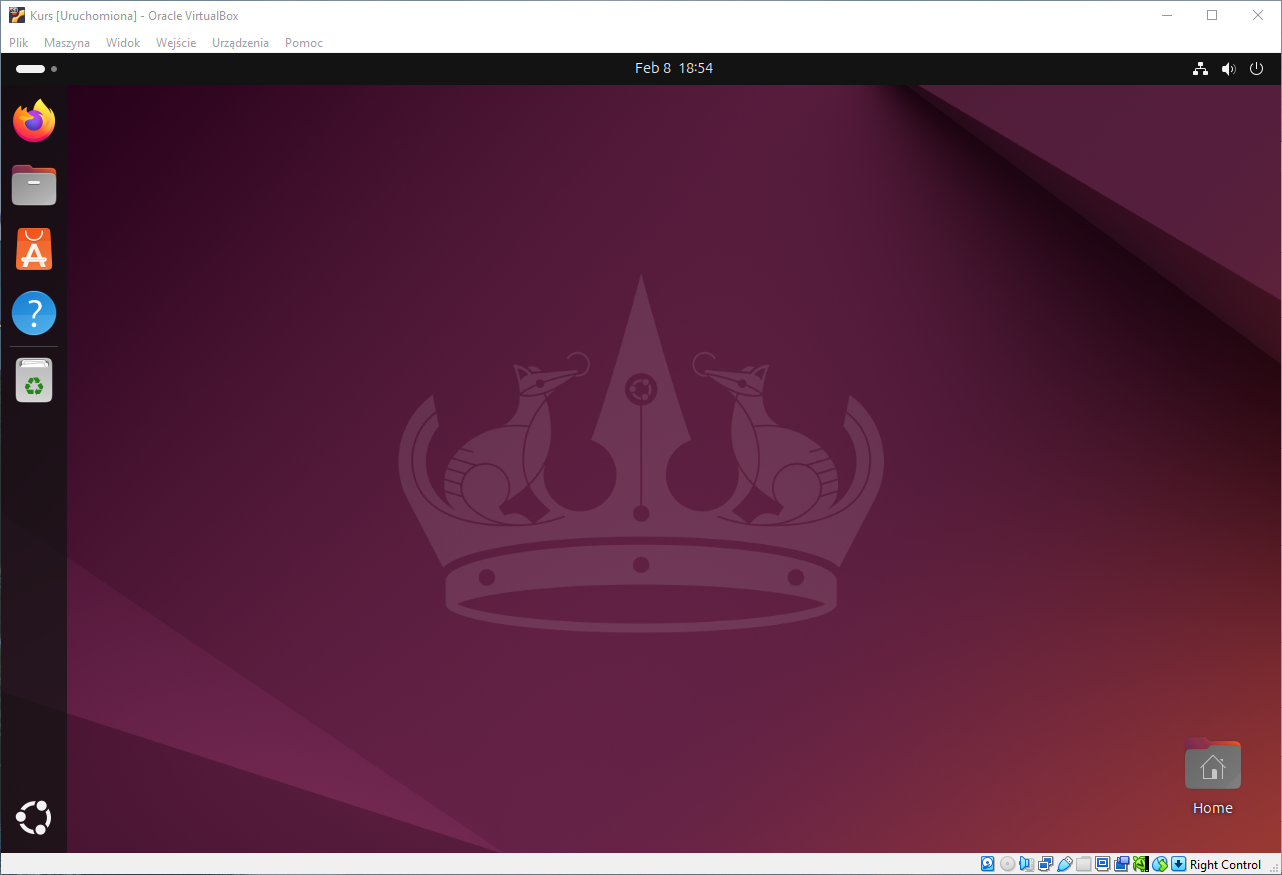
Rysunek 1.7 Okno wyboru skonfigurowanych maszyn wirtualnych

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.

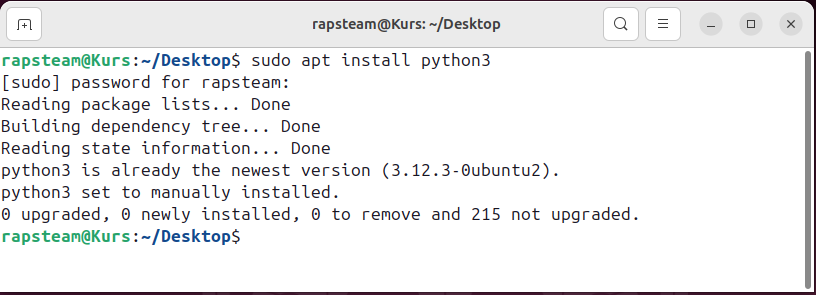


Rysunek 1.9 Pulpit systemu Ubuntu

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

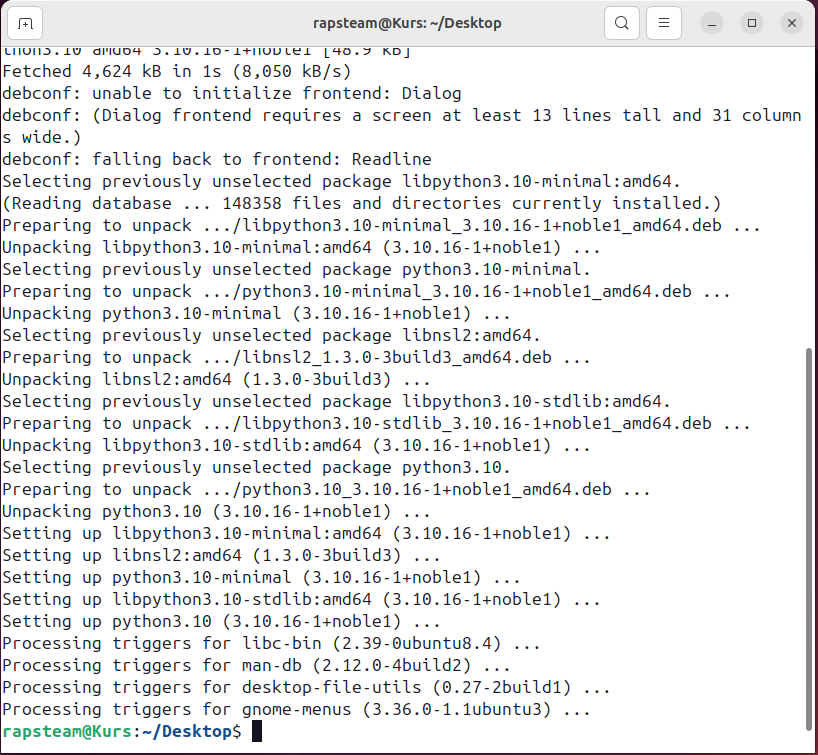
Możliwe jest także wskazanie konkretnej wersji pakietu Python do instalacji. Dzięki instalacji wielu wersji pakietu, można zapewnić kompatybilność różnych projektów, które wymagają specyficznych wersji interpretera i bibliotek. Pozwala to uniknąć konfliktów między zależnościami oraz zapewnia stabilność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

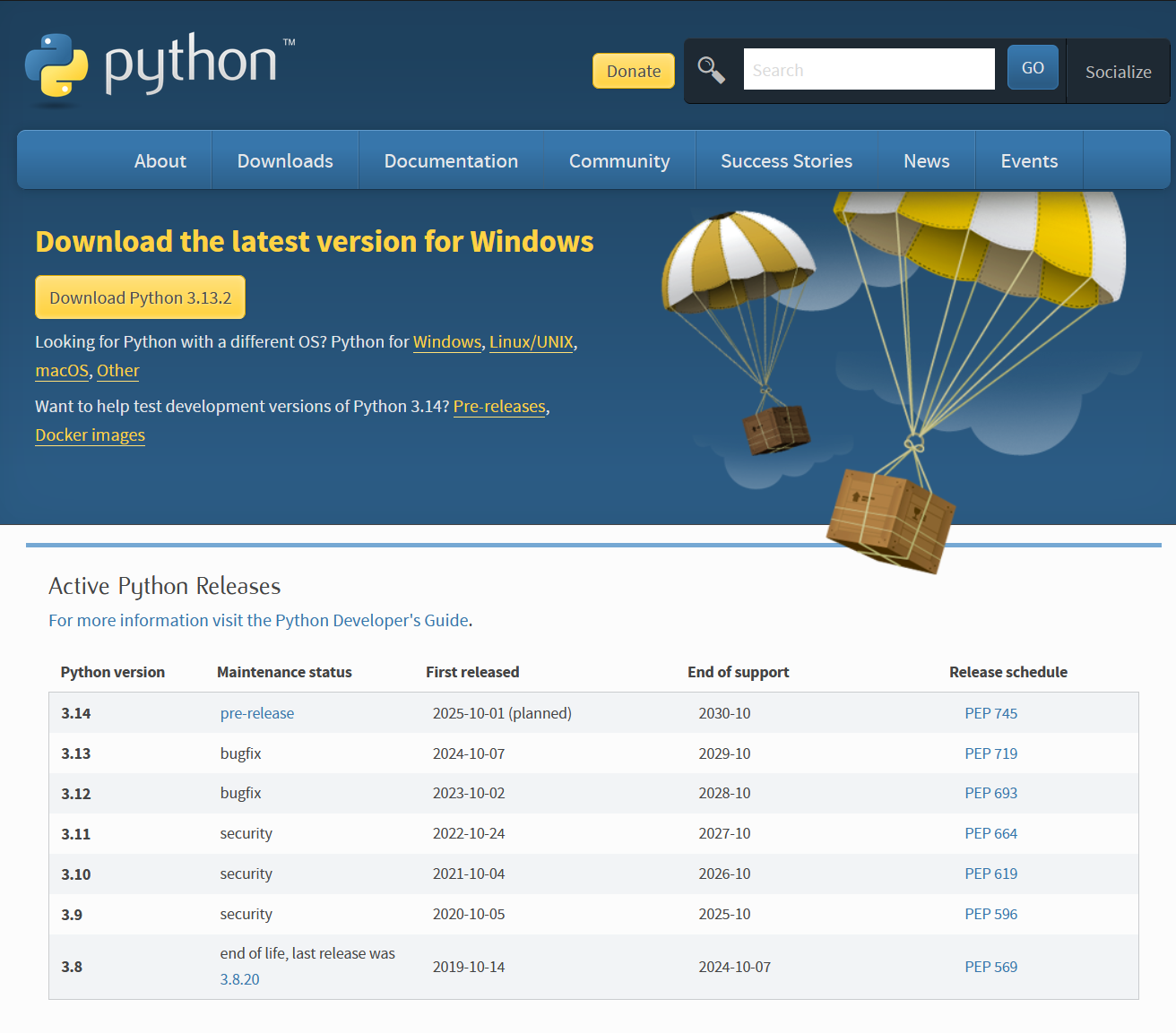


Rysunek 1.11. Proces instalacji pakietu Python w systemie Ubuntu

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:

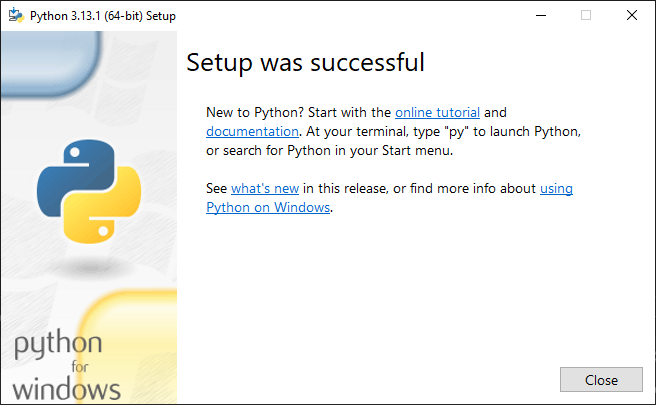
* Instalacja uproszczona (*Install Now*) – instaluje interpreter w domyślnej, wskazanej przez program lokalizacji systemowej oraz załącza do tego procesu szereg podstawowych, najpopularniejszych bibliotek dla tego języka, menadżer pakietów PIP oraz ustawia skojarzenie plików z rozszerzeniem *.py* jako 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.

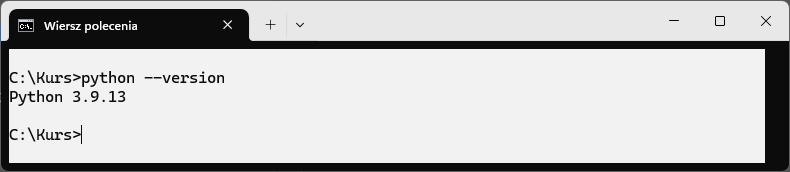
### Weryfikacja wersji interpretera języka Python

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 |



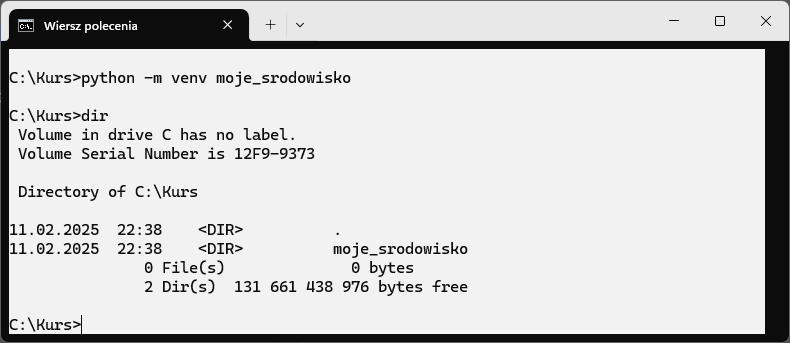
Rysunek 1.15. Weryfikacja instalacji Python w systemie Windows

### 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 |

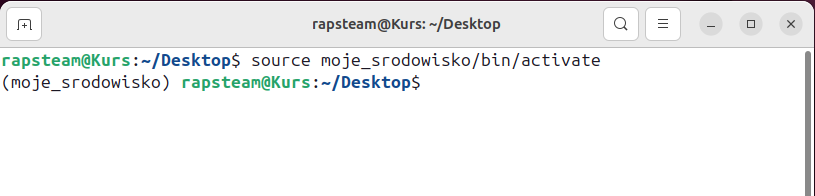


Rysunek 1.16. Tworzenie wirtualnego środowiska za pomocą pakietu venv

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 |

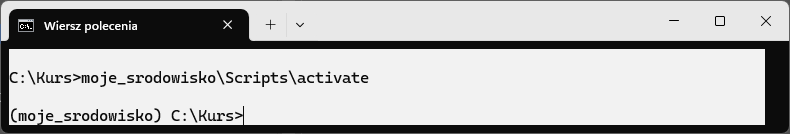


Rysunek 1.17. Wyłączenie środowiska wirtualnego w systemie Ubuntu

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 |



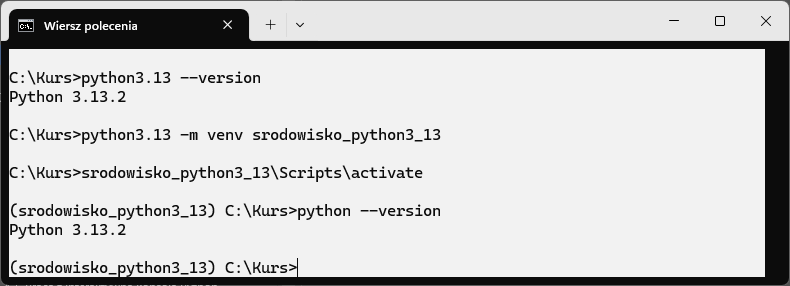
Rysunek 1.18. Efekt aktywacji środowiska wirtualnego (win)

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 1.20 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

|  |  |
| --- | --- |
| |  | | --- | | # Plik: example.py  # Importowanie modułu standardowego  **import** **datetime**  # Definicja funkcji  **def** **greet**(name):  """Funkcja zwracająca powitanie dla podanej osoby."""  **return** f"Cześć, {name}! Dzisiaj jest {datetime.date.today()}." # Wcięcia 4 spacje  # Wywołanie funkcji, jeśli skrypt jest uruchamiany bezpośrednio  **if** \_\_name\_\_ == "\_\_main\_\_":  print(greet("Patryk")) # Wcięcia 4 spacje  print("Koniec skryptu") # Wcięcia 0 spacji | |

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

|  |  |
| --- | --- |
| |  | | --- | | # Plik: break\_lines.py  # Przykład funkcji z długą nazwą  **def** **funkcja\_z\_dluga\_nazwa**(argument\_pierwszy, argument\_drugi,  argument\_trzeci, argument\_czwarty):  **return** argument\_pierwszy + argument\_drugi + argument\_trzeci + argument\_czwarty  wynik = funkcja\_z\_dluga\_nazwa(**1**, **2**, **3**, **4**)  print(wynik) | |

### 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!”

|  |  |
| --- | --- |
| |  | | --- | | # Plik: hello\_python.py  **def** **powitanie**():  **return** f"Witaj w świecie Pythona!"  print(powitanie()) | |

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

|  |  |
| --- | --- |
| |  | | --- | | # Plik: arguments.py  # Argumenty: 1 2 3  **import** **sys**  **if** len(sys.argv) > **1**:  print(f"Podano argumenty: {sys.argv[**1**:]}")  **else**:  print("Nie podano argumentów.") | |

### 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 1.21. 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 1.22. 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

|  |  |
| --- | --- |
| |  | | --- | | # Plik: hello.py  **def** **powitanie**(imie):  **return** f"Hello, {imie}!"  print(powitanie("Ania")) | |

**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

|  |  |
| --- | --- |
| |  | | --- | | # Plik: today\_date.py  # Przykład funkcji z długą nazwą  **from** **datetime** **import** datetime  # Pobranie aktualnej daty i godziny  dzisiaj = datetime.now().strftime("%Y-%m-%d %H:%M:%S")  # Zapis do pliku  **with** open("data\_log.txt", "a") **as** plik:  plik.write(f"{dzisiaj}**\n**")  print(f"Dzisiejsza data została zapisana: {dzisiaj}") | |

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.

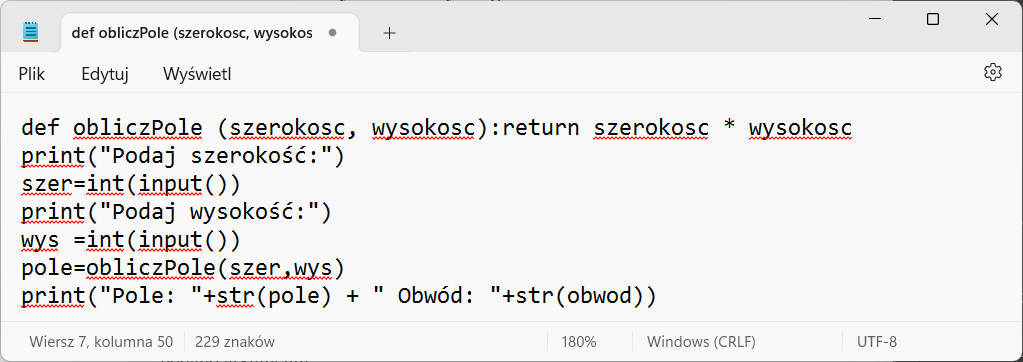
### Zadania podsumowujące temat

**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 uruchom 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**

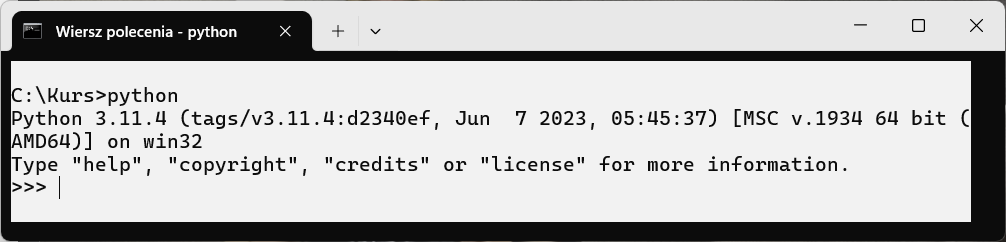
* Napisz skrypt, który pobiera aktualną datę i godzinę i zapisuje ją do pliku *log.txt*.
* Przykładowy format zapisu *2025-02-05 14:30:12 - Uruchomiono skrypt*
* Jeśli plik *log.txt* nie istnieje, skrypt powinien go utworzyć.

## 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.



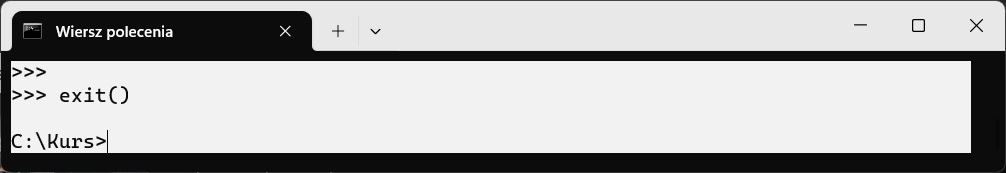
Rysunek 1.23. Uruchomienie trybu interaktywnego dla domyślnej wersji Python

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.

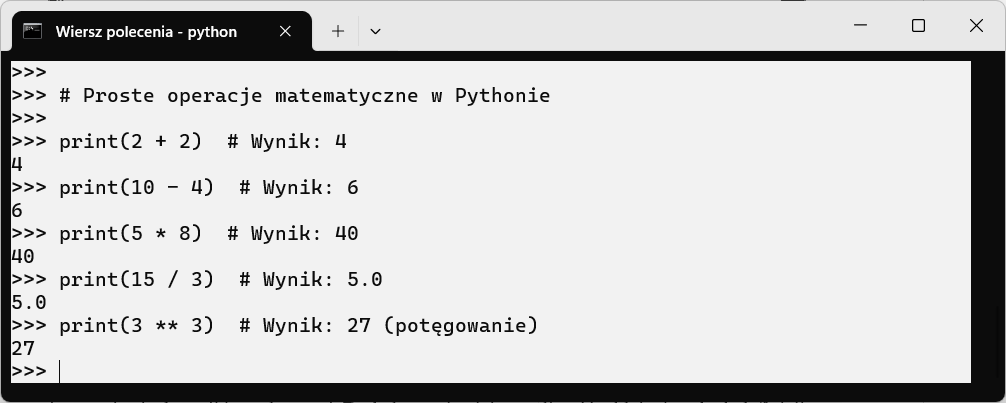


Rysunek 1.24. Wyłączenie trybu interaktywnego

### 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 1.25. Natychmiastowe obserwowanie wyników w trybie interaktywnym

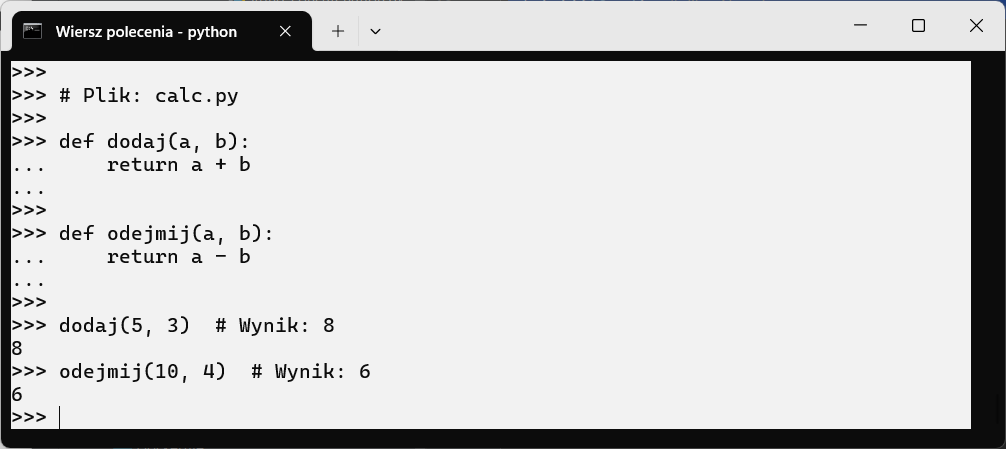
### Testowanie fragmentów kodu w trybie interaktywnym

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

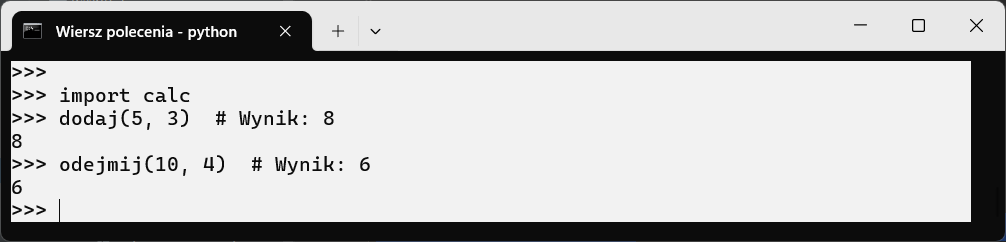
|  |  |
| --- | --- |
| |  | | --- | | # Plik: calc.py  **def** **dodaj**(a, b):  **return** a + b  **def** **odejmij**(a, b):  **return** a - b  print(dodaj(**5**, **3**))  print(odejmij(**10**, **4**)) | |

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 1.26. 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 1.27. Testowanie fragmentów kodu z pliku .py w konsoli

### Zadania podsumowujące temat

**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*
* Sprawdź działanie *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

|  |  |
| --- | --- |
| |  | | --- | | # Plik: arithmetic.py  print("Dodawanie:", **5** + **1**)  print("Odejmowanie:", **10** - **1**)  print("Mnożenie:", **5** \* **2**)  print("Dzielenie:", **10** / **3**)  print("Dzielenie całkowite:", **10** // **3**)  print("Modulo:", **10** % **3**)  print("Potęgowanie:", **2** \*\* **3**) | |

**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

|  |  |
| --- | --- |
| |  | | --- | | # Plik: abbreviated\_operators.py  x = **5**  x += **1** # To samo co: x = x + 1  print("Dodawanie (+=):", x)  y = **2**  y \*\*= **3** # To samo co: y = y \*\* 3  print("Potęgowanie (\*\*=):", y) | |

**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

|  |  |
| --- | --- |
| |  | | --- | | # Plik: math\_operations\_types.py  a = **5** # int  b = **2.5** # float  print("Dodawanie int + float:", a + b)  x = **10**  y = **4**  print("Dzielenie int / int:", x / y) # Wynik zawsze float  z = **3.0**  print("Mnożenie float \* int:", z \* **2**) # Wynik float | |

### 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

|  |  |
| --- | --- |
| |  | | --- | | # Plik: text\_case\_methods.py  tekst = "python jest świetny!"  print("Wielkie litery:", tekst.upper()) # PYTHON JEST ŚWIETNY!  print("Małe litery:", tekst.lower()) # python jest świetny!  print("Tytułowe litery:", tekst.title()) # Python Jest Świetny!  print("Pierwsza litera wielka:", tekst.capitalize()) # Python jest świetny! | |

**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

|  |  |
| --- | --- |
| |  | | --- | | # Plik: string\_repetition.py  # Powielanie napisu za pomocą operatora \*  print("Powielanie (\*):", "Python! " \* **3**)  # Powielanie pojedynczego znaku  print("Powielanie znaku:", "=" \* **20**)  # Powielanie z użyciem join()  print("Powielanie (join()):", "-".join(["Hello"] \* **3**)) | |

**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

|  |  |
| --- | --- |
| |  | | --- | | # Plik: text\_case\_methods.py  tekst = "python jest świetny!"  print("Wielkie litery:", tekst.upper()) # PYTHON JEST ŚWIETNY!  print("Małe litery:", tekst.lower()) # python jest świetny!  print("Tytułowe litery:", tekst.title()) # Python Jest Świetny!  print("Pierwsza litera wielka:", tekst.capitalize()) # Python jest świetny! | |

### 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.

|  |  |
| --- | --- |
| |  | | --- | | # Plik: len\_type\_demo.py  # Użycie len()  tekst = "Python"  lista = [**10**, **20**, **30**, **40**]  print("Długość napisu:", len(tekst)) # Wynik: 6  print("Liczba elementów w liście:", len(lista)) # Wynik: 4  # Użycie type()  liczba = **42**  napis = "Hello"  zbior = {**1**, **2**, **3**}  print("Typ zmiennej liczba:", type(liczba)) # Wynik: <class 'int'>  print("Typ zmiennej napis:", type(napis)) # Wynik: <class 'str'>  print("Typ zmiennej zbior:", type(zbior)) # Wynik: <class 'set'> | |

### Zadania podsumowujące temat

**Zadanie 1.5.1. Operacje arytmetyczne**

* Napisz program, który pobierze od użytkownika dwie liczby, a następnie obliczy i wyświetli ich sumę, różnicę, iloczyn oraz wynik dzielenia całkowitego.

**Zadanie 1.5.2. Manipulacja napisami**

* Napisz program, który pobierze od użytkownika imię, nazwisko.
* Połączy wszystkie podane dane w jeden napis, używając operatora +.
* Wyświetli wynik w trzech formatach:
  + **Za pomocą +**: imię nazwisko
  + **Za pomocą *join()***: imię nazwisko
  + **Wielkimi literami:** IMIĘ NAZWISKO

*\* Wykorzystaj .upper() do zmiany liter na wielkie.*

**Zadanie 1.5.3. Funkcje wbudowane**

* Napisz program, który pobierze od użytkownika liczbę, a następnie:
  + Wyświetli jej typ (type()).
  + Przekonwertuje ją na int, float i str, wyświetlając wyniki.

**Zadanie 1.5.4. \* Formatowanie danych użytkownika II**

* Napisz program, który poprosi użytkownika o wpisanie liczb oddzielonych spacją, a następnie:
  + Zamieni je na listę liczb całkowitych.
  + Obliczy i wyświetli ich sumę (*sum()*) oraz liczbę elementów (l*en()*).

**Zadanie 1.5.5. \* Formatowanie danych użytkownika II**

* Pobierze od użytkownika imię, nazwisko i wiek.
* Połączy imię i nazwisko w jeden napis (użyj join()).
* Wyświetli imię i nazwisko w wielkich literach oraz formacie tytułowym.
* Sprawdzi długość nazwiska (len()) i typ wieku (type()).

## 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 |

Wiedząc już jak w języku Python tworzy się zmienne, warto powiedzieć czym różni się on od innych języków programowania. 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 kilka typy danych, które można podzielić na kilka głównych kategorii.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 jezyka 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 1.28 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.*



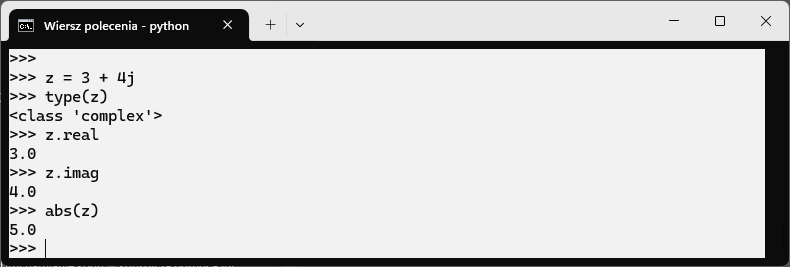
Rysunek 1.29 Przykłady liczb zmiennoprzecinkowych



Rysunek 1.30 Przykład błędu zaokrąglenia liczb zmiennoprzecinkowych

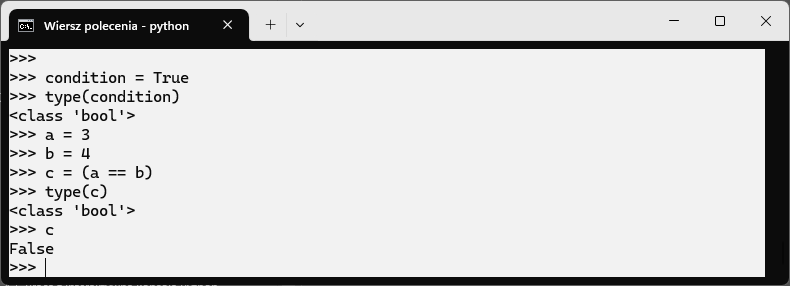
**Liczby zespolone** **(complex)**

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 1.31 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 1.32 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)**

W jęzku Python to rodzaj danych służący do reprezentacji ciągów znaków. Obiekty tego typu, 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 *string*, 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. Za pomocą funkcji *str()* można dokonywać konwersji wielu typów zmiennych które są dostępne w języku Python, a najczęściej będą to *float, int* i *bool*

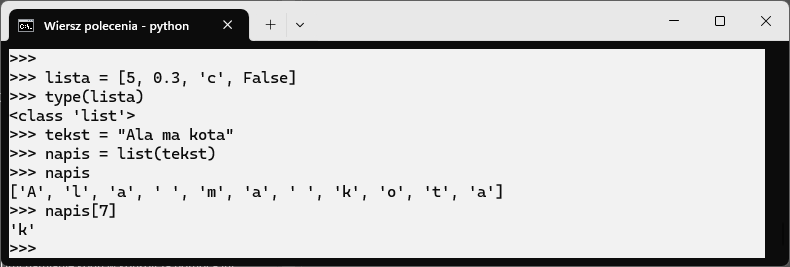


Rysunek 1.33 Przykład stworzenia ciągu znaków w postaci zdania, oraz pojedynczego znaku

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

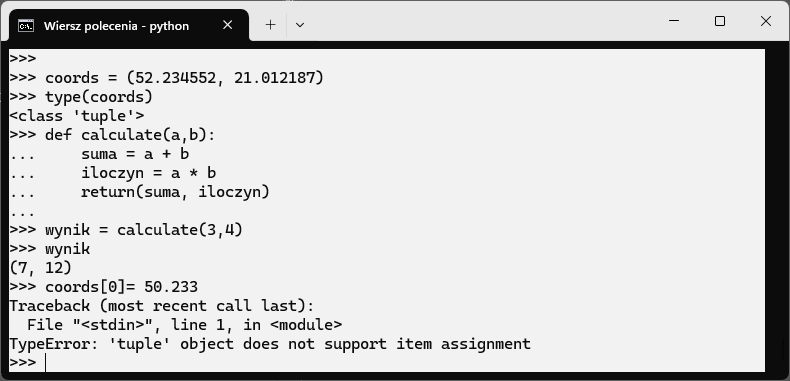
**Listy (list)**

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 1.34 Przykład utworzenia listy zawierającej wiele typów danych

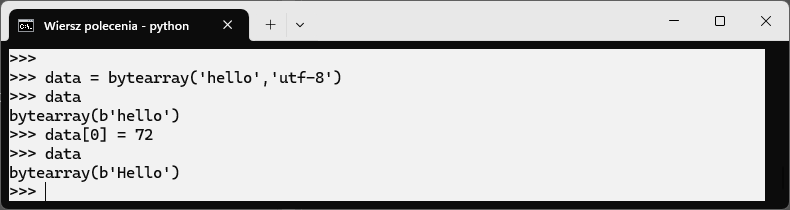
**Krotki (tuple)**  
Krotki to sekwencje, podobne do list, ale raz utworzone nie można ich zmienić. Oznacza to, że dane zapisane w krotce pozostają stałe, co jest przydatne, gdy potrzebujemy pewności, że zestaw informacji nie zostanie przypadkowo zmodyfikowany. Aby utworzyć krotkę, wystarczy umieścić 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 1.35 Niezmienność krotek w języku Python

**Sekwencja bajtów (bytes/bytearray)**

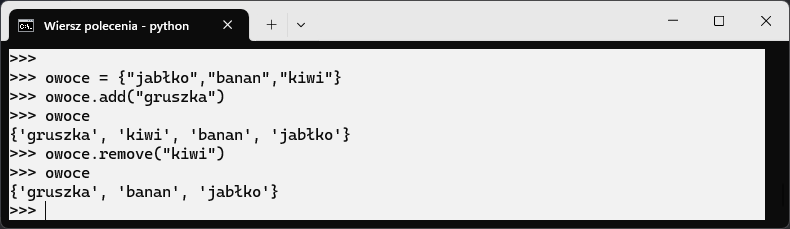
Sekwencyjne typy reprezentujące dane binarne. Typ *bytes* jest niemutowalnym ciągiem liczb całkowitych, gdzie każda liczba mieści się w przedziale od 0 do 255, i jest wykorzystywany do reprezentowania danych binarnych. Wersja mutowalna, *bytearray*, umożliwia modyfikację zawartości po jej utworzeniu,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



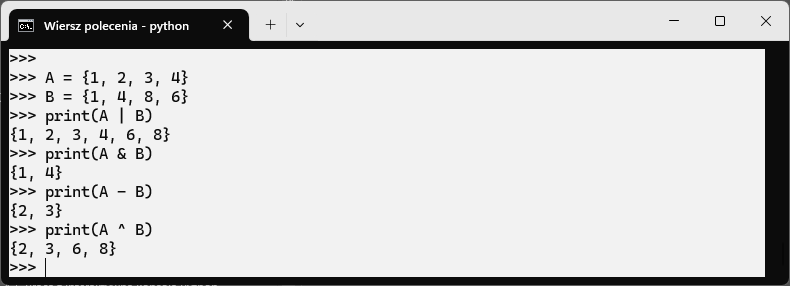
Rysunek 1.36 Przykład utworzenia krotki oraz stowrzenia jej jako zwracany typ funkcji

### Zbiory

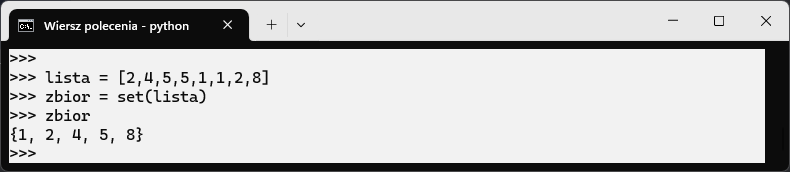
Zbiór tworzony jest poprzez ujęcie elementów w nawiasy klamrowe, na przykład: *{1, 2, 3},* lub za pomocą funkcji *set(),* która konwertuje iterowalne obiekty na zbiór. Zbiór umożliwia wykonywanie operacji matematycznych, takich jak suma, różnica, przecięcie czy symetryczna różnica, co przydaje się przy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



Rysunek 1.37. Operacja utworzenia zbioru i dodania i usunięcia jego elementu



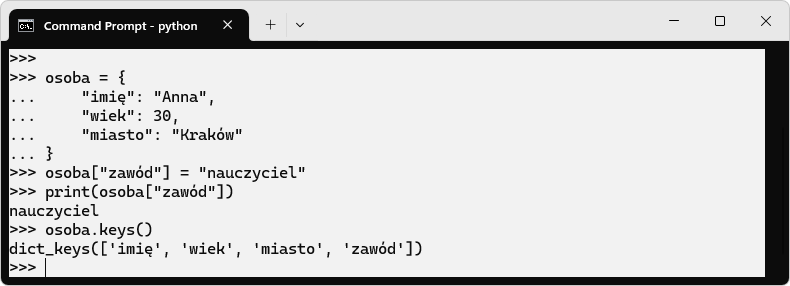
Rysunek 1.38. Podstawowe operacje na zbiorach



Rysunek 1.39 Usuwanie duplikatów i sortowanie elementów listy poprzez rzutowanie

### Typ mapujący

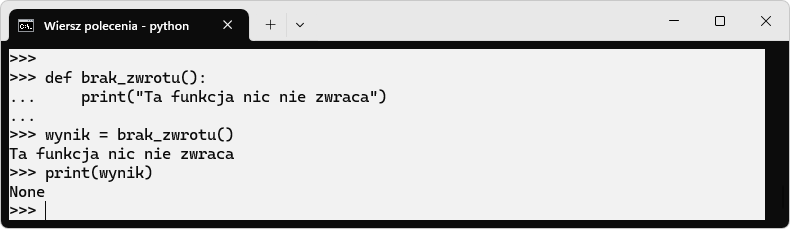
Słownik (dictionary) to jeden z typów mapujących języka Python, oparty na unikalnych kluczach niemutowanego typu, które są przypisane do odpowiadających im wartości. Elementy słownika zapisuje się w nawiasach klamrowych w formie par klucz-wartość, oddzielonych dwukropkiem. Słowniki umożliwiają szybki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



Rysunek 1.40 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 1.41 Domyślny zwrot None w Pythonie

### Zadania

**Zadanie 1.6.1. Sprawdzenie typów utworzonych zmiennych**

* Utwórz po jednej zmiennej z każdego typów i wyświetl jej zawartość.

**Zadanie 1.6.2.**

* Dokonaj konwersji typów. Zamień:
  + Zmienną typów *str* na *int, float, bool* i *complex*
  + Zmienną typu *float*  na *str, int, bool* i *complex*

**Zadanie 1.6.3. \***

* Utwórz zmienne *x = 5* i *y = 3.14*
* Wykonaj operację:
  + *x + y*  i zobacz co zwróci. Jaki to będzie typ?
  + *str(x) + str(y)*. Co się stanie po takiej operacji?

**Zadanie 1.6.4. \***

* Utwórz listę *list = [1, 2, 3]*
* Zmień drugi element na *5*
* Utwórz krotkę *krotka = (4,5,6)*
* Spróbuj zmienić drugi element na *7.* Co się stanie?

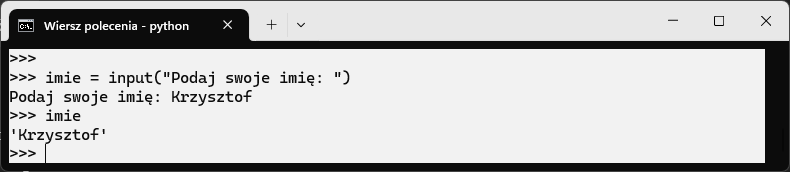
**Zadanie 1.6.5.**

* Utwórz zmienną *produkt* będącą słownikiem
* Niech zmienna posiada pola: *nazwa, cena* i *ilość* a następnie wypełnij je przykładowymi wartościami
* Wyświetl zawartość tej zmiennej

## Używanie metod wejścia i wyjścia

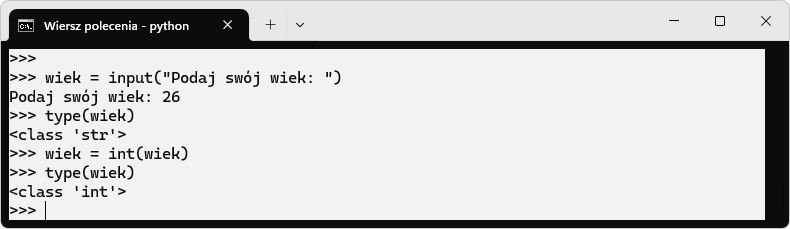
### Funkcja input()

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 1.42 Pobranie od użytkownika danych za pomocą funkcji input()

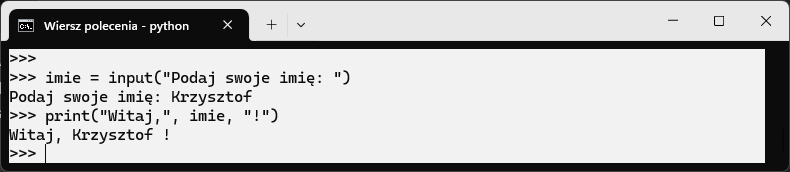
Warto zauważyć, że funkcja *input()* zawsze zwraca wartość typu string, co oznacza, że przy pobieraniu liczb konieczna jest konwersja, na przykład przy użyciu funkcji *int()* lub *float()*:



Rysunek 1.43 Pobranie informacji od użytkownika i konwersja jej typu z str na int

### Funkcja print()

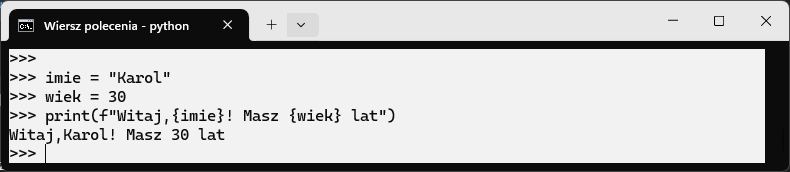
Funkcja *print()* służy do prezentowania wyników na ekranie poprzez standardowe wyjście. Argumenty przekazywane do funkcji są konwertowane na ciągi znaków, a następnie wyświetlane. Domyślnie kolejne argumenty oddzielane są spacją, a na końcu wyniku funkcja dodaje znak nowej linii. Przykładowe użycie 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 1.44 Pobranie informacji od użytkownika i wyświetlenie jej za pomocą funkcji print()

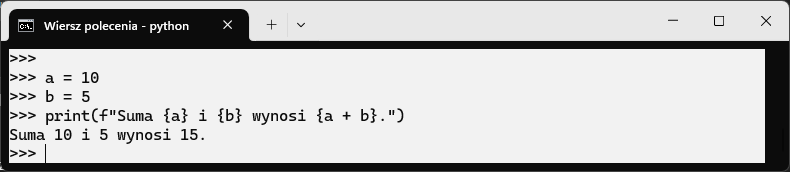
### Formatowanie stringów

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 1.45 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 1.46 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]."*

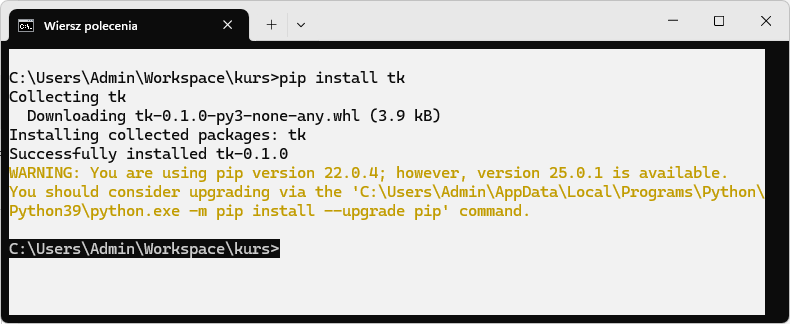
## Korzystanie z bibliotek i menadżera PIP

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*).

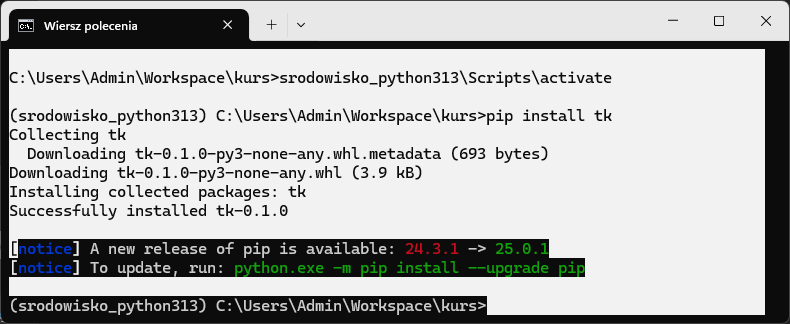
### Instalacja i wykorzystanie bibliotek z pomocą narzędzia PIP

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.



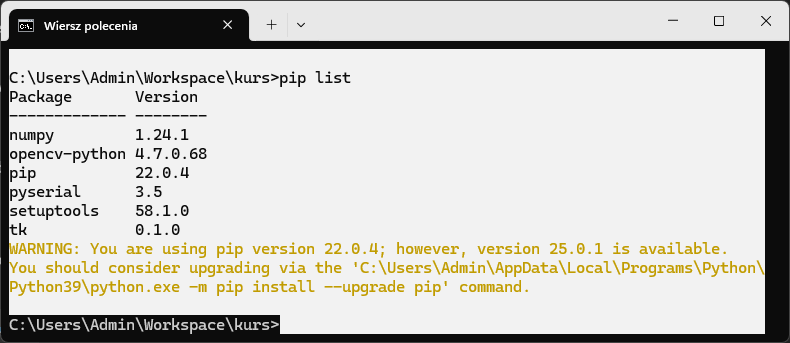
Rysunek 1.47 Przykład instalacji biblioteki tkinter na poziomie globalnym

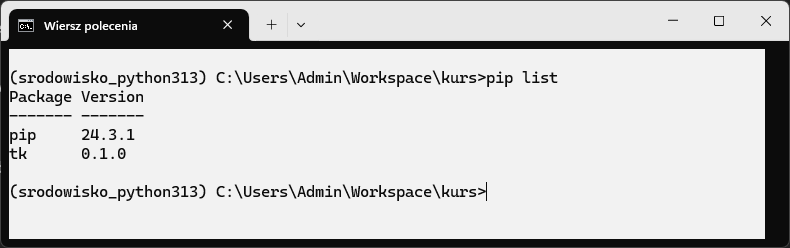
Instalowanie bibliotek za jego pomocą jest bardzo wygodne i szybkie. Należy jednak mieć na uwadze fakt, że wykonywanie go na poziomie systemu, powoduje, że biblioteka jest instalowana globalnie i może to powodować konflikty z innymi modułami korzystającymi z innych zależności. Dlatego też, przy dużych 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 1.48 Przykład instalacji biblioteki tkinter wewnątrz środowiska wirtualnego

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





Rysunek 1.49 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

|  |  |
| --- | --- |
| |  | | --- | | # Plik: turtle\_square.py  # where | |

Listing 1.19. Rysowanie okręgu za pomocą biblioteki

|  |
| --- |
| [module\_1\_lesson\_4\_turtle\_circle.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 modułu

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 moduł

**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 \***

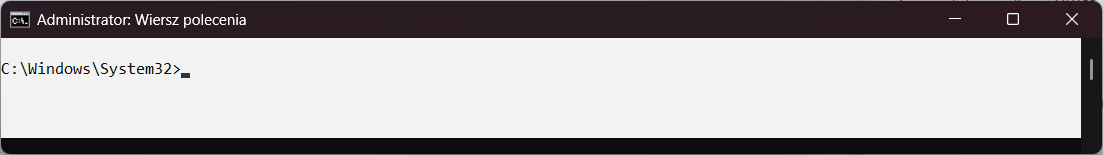
Napisz program, który

* 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.**

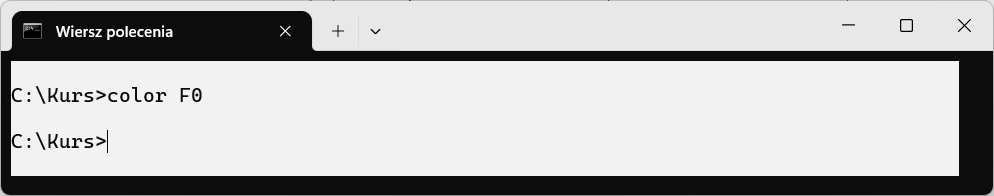
Jedną z metod uruchomienia konsoli z prawami administratora jest wpisanie *CMD* w oknie wyszukiwania systemu Windows, wybranie prawym przyciskiem myszy *Wiersza Poleceń*, a następnie *Uruchom jako administrator*.



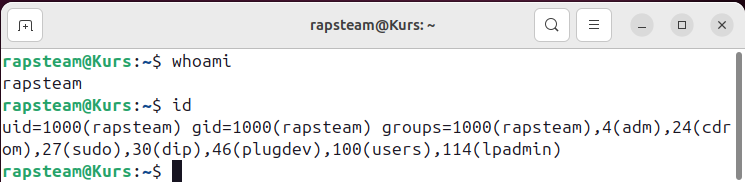
**Zadanie 1.1.2.**

* 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.**

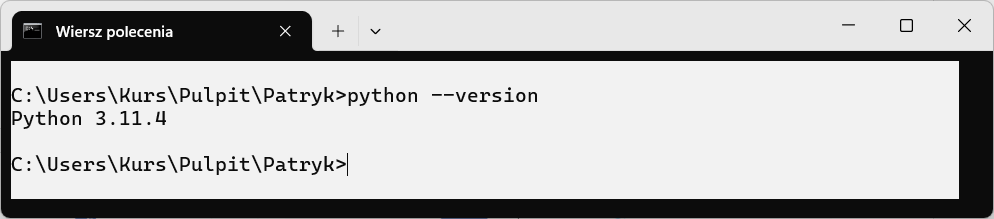
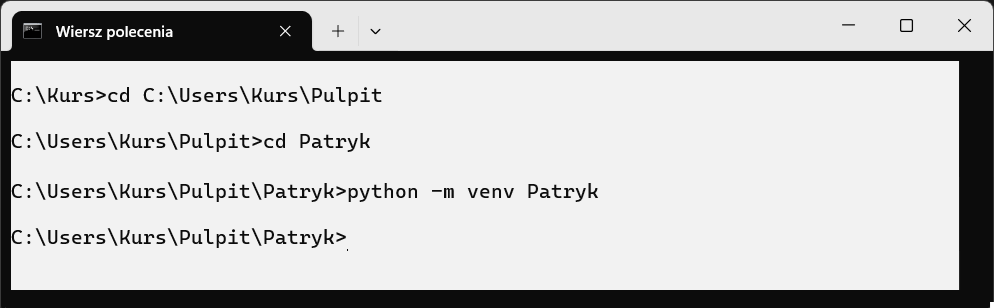
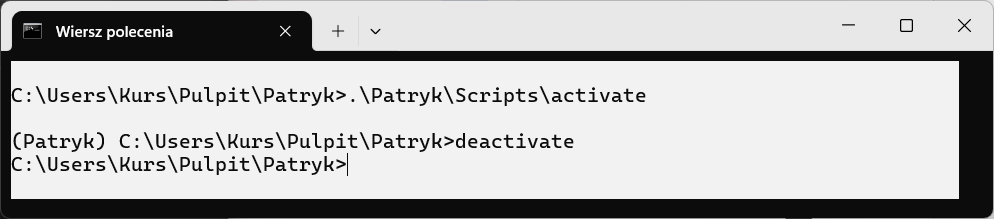


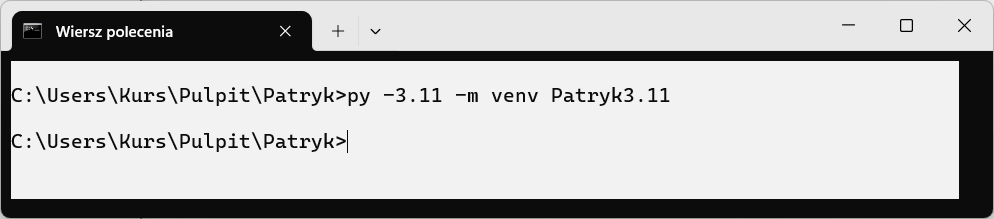
**Zadanie 1.1.4. \***

I

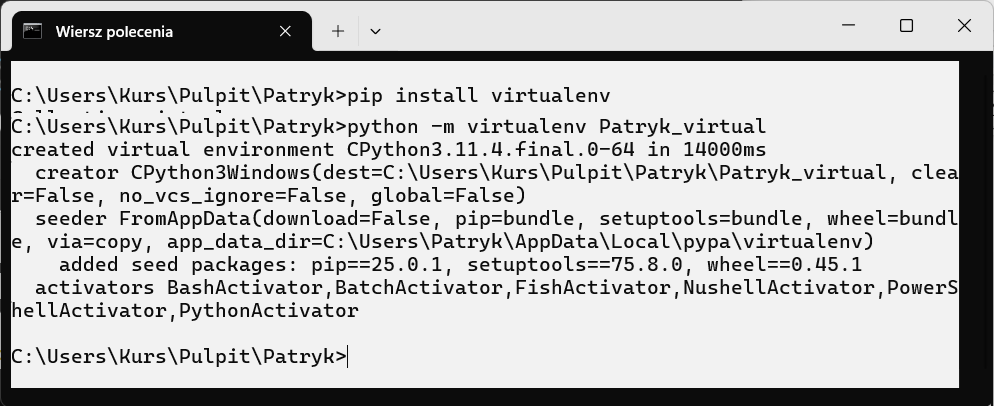
**Zadanie 1.1.5. \***

|  |  |
| --- | --- |
| |  | | --- | | **import** **subprocess**  # Polecenie CMD do uruchomienia konsoli, zmiany koloru i wyświetlenia wersji Pythona  cmd\_command = 'start cmd /k "color F0 && python --version"'  # Uruchomienie konsoli CMD z podanym poleceniem  subprocess.run(cmd\_command, shell=**True**) | |

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


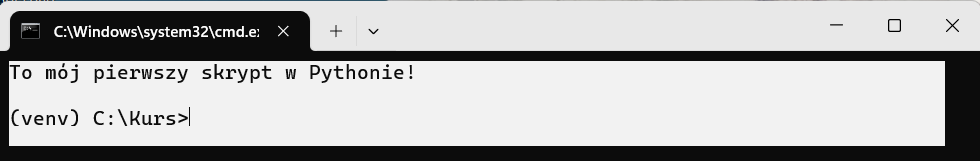
**Zadanie 1.2.4. \***   


**Zadanie 1.2.5. \***



**Zadanie 1.3.1.**

|  |  |
| --- | --- |
| |  | | --- | | print("To mój pierwszy skrypt w Pythonie!") | |

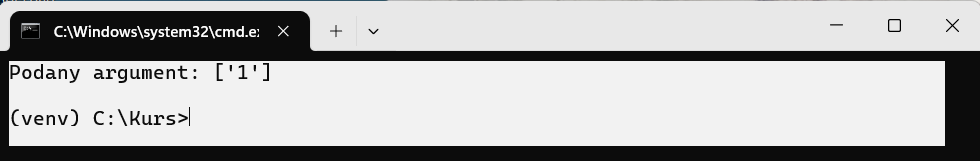


**Zadanie 1.3.2.**

|  |  |
| --- | --- |
| |  | | --- | | # Input: 2,3  **def** **obliczPole**(szerokosc, wysokosc):  **return** szerokosc \* wysokosc  print("Podaj szerokość:")  szer = int(input())  print("Podaj wysokość:")  wys = int(input())  pole = obliczPole(szer, wys)  wynik = **2** \* (szer + wys) # Dodana obliczeniowa obwodu  print("Pole:", pole, "Obwód:", wynik) | |

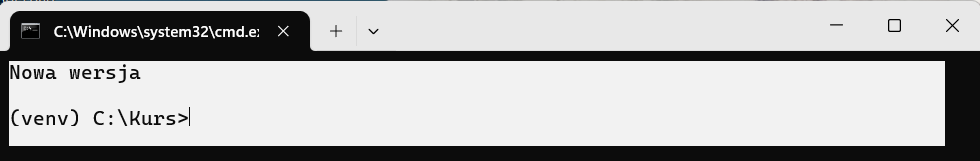
**Zadanie 1.3.3.**

|  |  |
| --- | --- |
| |  | | --- | | # Argumenty: 1  **import** **sys**  # Sprawdzenie, czy podano argument  **if** len(sys.argv) > **1**:  argument = sys.argv[**1**]  print(f"Podany argument: {argument}")  **else**:  print("Nie podano argumentu!") | |



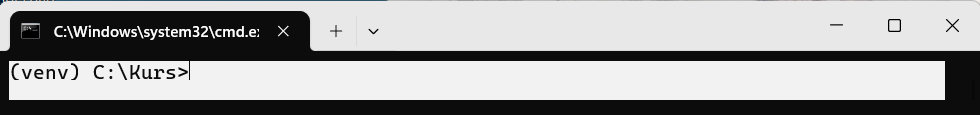
**Zadanie 1.3.4. \***

|  |  |
| --- | --- |
| |  | | --- | | **import** **sys**  **if** sys.version\_info < (**3**, **9**):  print("Stara wersja")  **else**:  print("Nowa wersja") | |

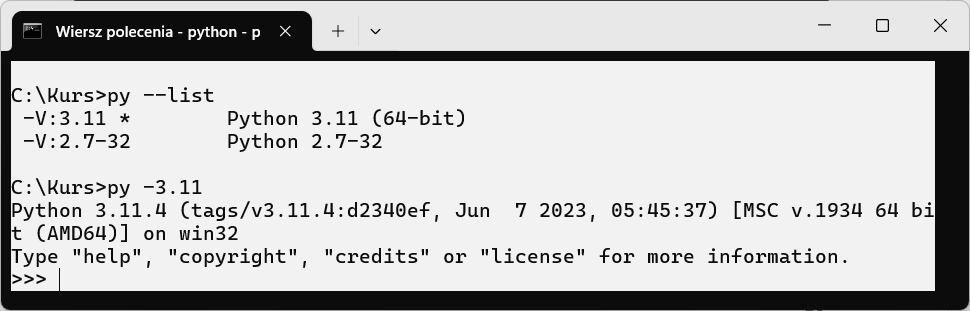


**Zadanie 1.3.5. \***

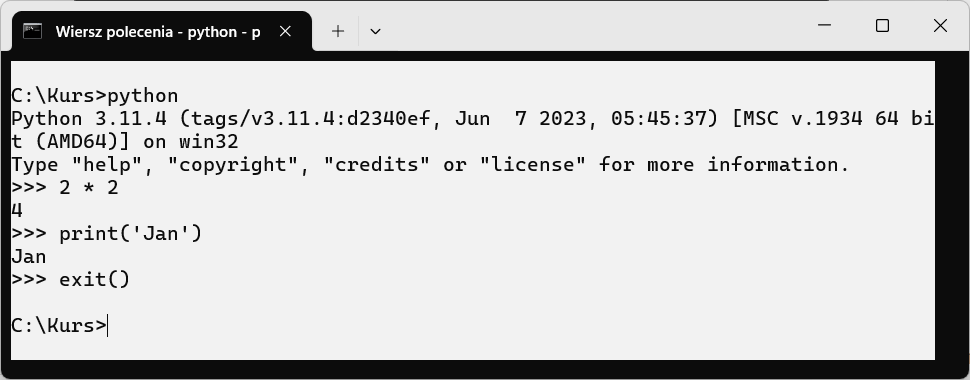
|  |  |
| --- | --- |
| |  | | --- | | **from** **datetime** **import** datetime  # Pobranie aktualnej daty i godziny w formacie YYYY-MM-DD HH:MM:SS  current\_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")  # Treść wpisu do loga  log\_entry = f"{current\_time} - Uruchomiono skrypt**\n**"  # Nazwa pliku loga  log\_file = "log.txt"  # Otwarcie pliku w trybie dopisywania (lub utworzenie, jeśli nie istnieje)  **with** open(log\_file, "a", encoding="utf-8") **as** file:  file.write(log\_entry)  print(f"Zapisano do {log\_file}: {log\_entry.strip()}") | |



**Zadanie 1.4.1.**



**Zadanie 1.4.2.**



**Zadanie 1.4.3.**

|  |  |
| --- | --- |
| |  | | --- | | # Zadanie: 1\_4\_3.py  **import** **math**  print(math.pi) | |

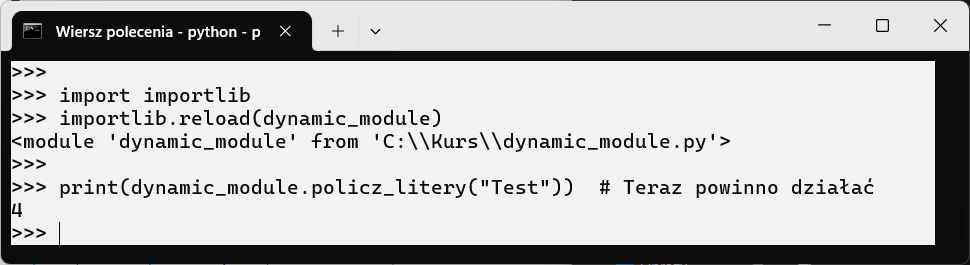
**Zadanie 1.4.4. \***

|  |  |
| --- | --- |
| |  | | --- | | # Argumenty: 1 2 3  **import** **sys**  # Pobranie argumentów (pomijając nazwę skryptu)  arguments = sys.argv[**1**:]  # Sprawdzenie, czy podano argumenty  **if** arguments:  print(f"Podano {len(arguments)} argumentów: {', '.join(arguments)}")  **else**:  print("Nie podano argumentów!") | |

**Zadanie 1.4.5. \***

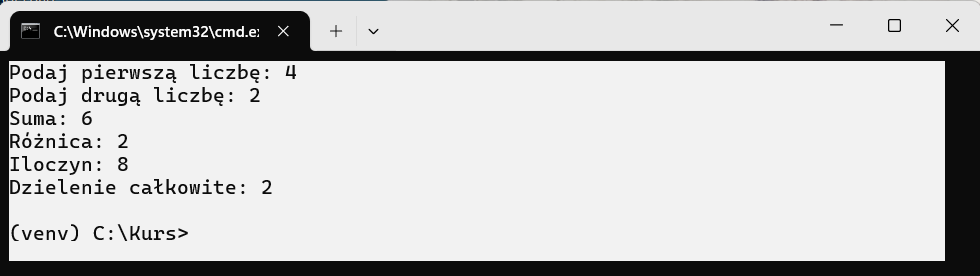
Funkcja *policz\_litery* zadziała dopiero po ponowny załadowaniu biblioteki poprzez *importlib.reload(dynamic\_module).*

|  |  |
| --- | --- |
| |  | | --- | | # dynamic\_module.py  **def** **powitaj**(imie):  **return** f"Cześć, {imie}!"  **def** **policz\_litery**(tekst):  **return** len(tekst) | |



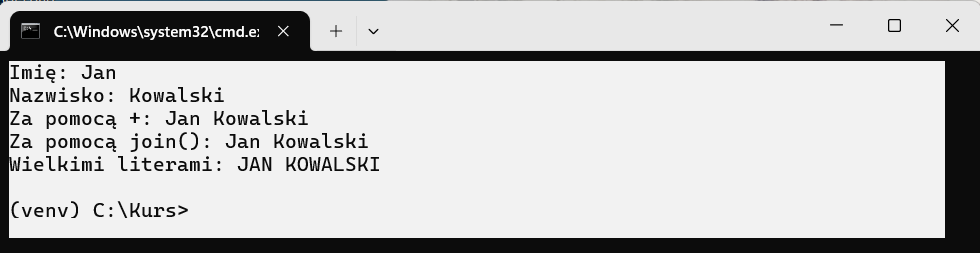
**Zadanie 1.5.1.**

|  |  |
| --- | --- |
| |  | | --- | | # Input: 4,2  # Pobranie dwóch liczb od użytkownika  a = int(input("Podaj pierwszą liczbę: "))  b = int(input("Podaj drugą liczbę: "))  # Wyświetlenie wyników  print(f"Suma: {a + b}")  print(f"Różnica: {a - b}")  print(f"Iloczyn: {a \* b}")  print(f"Dzielenie całkowite: {a // b **if** b != **0** **else** 'Nie można dzielić przez zero!'}") | |



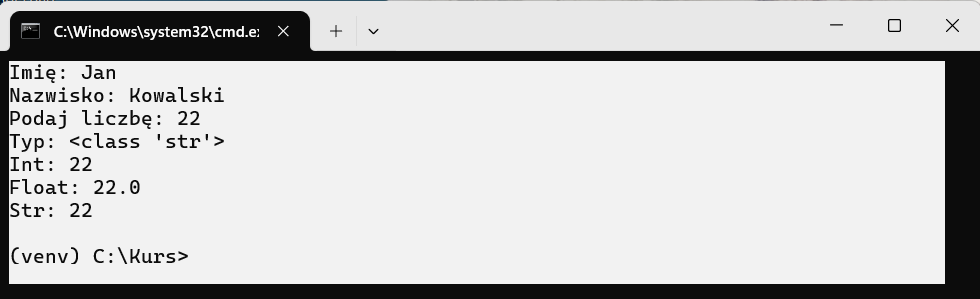
**Zadanie 1.5.2.**

|  |  |
| --- | --- |
| |  | | --- | | # Input: Jan,Kowalski  imie = input("Imię: ")  nazwisko = input("Nazwisko: ")  print("Za pomocą +:", imie + " " + nazwisko)  print("Za pomocą join():", " ".join([imie, nazwisko]))  print("Wielkimi literami:", (imie + " " + nazwisko).upper()) | |



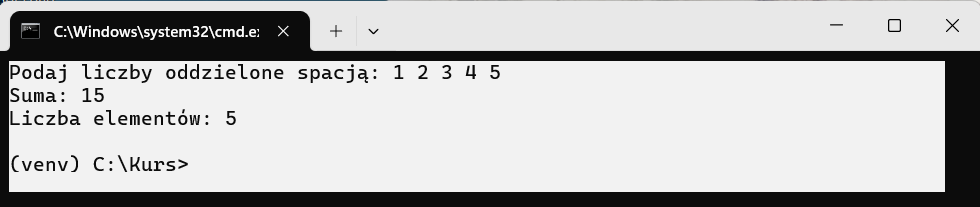
**Zadanie 1.5.3.**

|  |  |
| --- | --- |
| |  | | --- | | # Input: Jan,Kowalski,22  imie = input("Imię: ")  nazwisko = input("Nazwisko: ")  x = input("Podaj liczbę: ")  print(f"Typ: {type(x)}"  f"**\n**Int: {int(float(x))}"  f"**\n**Float: {float(x)}"  f"**\n**Str: {str(x)}") | |



**Zadanie 1.5.4. \***

|  |  |
| --- | --- |
| |  | | --- | | # Input: 1 2 3 4 5  # Pobranie liczb jako jednego ciągu znaków  liczby = input("Podaj liczby oddzielone spacją: ")  # Konwersja na listę liczb całkowitych  lista\_liczb = list(map(int, liczby.split()))  # Obliczenia  suma\_liczb = sum(lista\_liczb)  liczba\_elementow = len(lista\_liczb)  # Wyświetlenie wyników  print(f"Suma: {suma\_liczb}**\n**Liczba elementów: {liczba\_elementow}") | |



**Zadanie 1.5.5. \***

|  |  |
| --- | --- |
| |  | | --- | | # Input: Jan,Kowalski,18  imie = input("Imię: ")  nazwisko = input("Nazwisko: ")  wiek = input("Wiek: ")  pelne\_imie = " ".join([imie, nazwisko])  print(pelne\_imie.upper(), pelne\_imie.title(), len(nazwisko), type(wiek), sep="**\n**") | |

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

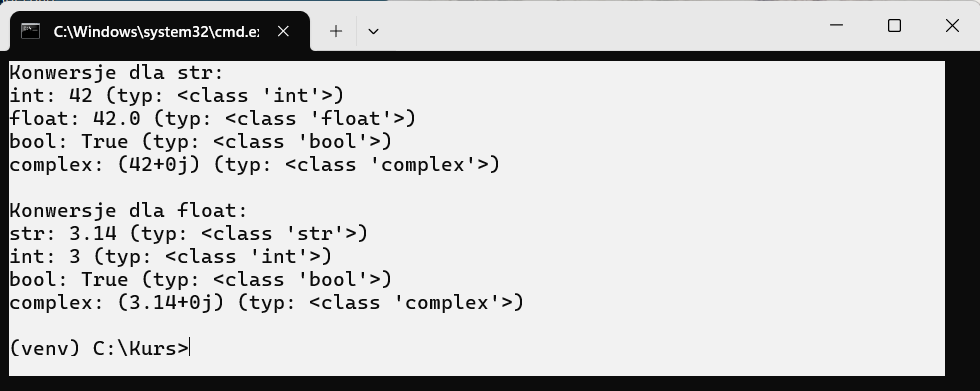
**Zadanie 1.6.1.**

|  |  |
| --- | --- |
| |  | | --- | | # Zadanie: 1\_6\_1.py  int\_value = **42**  float\_value = **3.14**  complex\_value = **1** + **2**j  bool\_value = **True**  str\_value = "Hello, World!"  list\_value = [**1**, **2**, **3**]  tuple\_value = (**4**, **5**, **6**)  bytes\_value = b"byte\_data"  set\_value = {**1**, **2**, **3**}  dict\_value = {"key1": "value1", "key2": "value2"}  none\_value = **None**  print("Typy liczbowe:")  print(f"int\_value: {int\_value} (typ: {type(int\_value)})")  print(f"float\_value: {float\_value} (typ: {type(float\_value)})")  print(f"complex\_value: {complex\_value} (typ: {type(complex\_value)})**\n**")  print("Typ logiczny:")  print(f"bool\_value: {bool\_value} (typ: {type(bool\_value)})**\n**")  print("Typy sekwencyjne:")  print(f"str\_value: {str\_value} (typ: {type(str\_value)})")  print(f"list\_value: {list\_value} (typ: {type(list\_value)})")  print(f"tuple\_value: {tuple\_value} (typ: {type(tuple\_value)})")  print(f"bytes\_value: {bytes\_value} (typ: {type(bytes\_value)})**\n**")  print("Typ zbiorowy:")  print(f"set\_value: {set\_value} (typ: {type(set\_value)})**\n**")  print("Typ mapujący:")  print(f"dict\_value: {dict\_value} (typ: {type(dict\_value)})**\n**")  print("Typ braku wartości:")  print(f"none\_value: {none\_value} (typ: {type(none\_value)})") | |



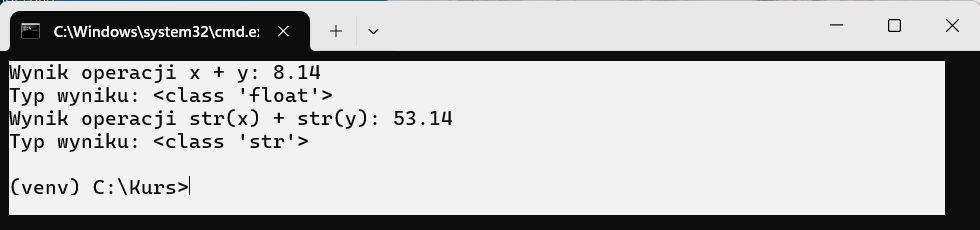
**Zadanie 1.6.2.**

|  |  |
| --- | --- |
| |  | | --- | | # Zadanie: 1\_6\_2.py  string\_value = "42"  int\_from\_str = int(string\_value)  float\_from\_str = float(string\_value)  bool\_from\_str = bool(string\_value)  complex\_from\_str = complex(string\_value)  print("Konwersje dla str:")  print(f"int: {int\_from\_str} (typ: {type(int\_from\_str)})")  print(f"float: {float\_from\_str} (typ: {type(float\_from\_str)})")  print(f"bool: {bool\_from\_str} (typ: {type(bool\_from\_str)})")  print(f"complex: {complex\_from\_str} (typ: {type(complex\_from\_str)})**\n**")  float\_value = **3.14**  str\_from\_float = str(float\_value)  int\_from\_float = int(float\_value)  bool\_from\_float = bool(float\_value)  complex\_from\_float = complex(float\_value)  print("Konwersje dla float:")  print(f"str: {str\_from\_float} (typ: {type(str\_from\_float)})")  print(f"int: {int\_from\_float} (typ: {type(int\_from\_float)})")  print(f"bool: {bool\_from\_float} (typ: {type(bool\_from\_float)})")  print(f"complex: {complex\_from\_float} (typ: {type(complex\_from\_float)})") | |



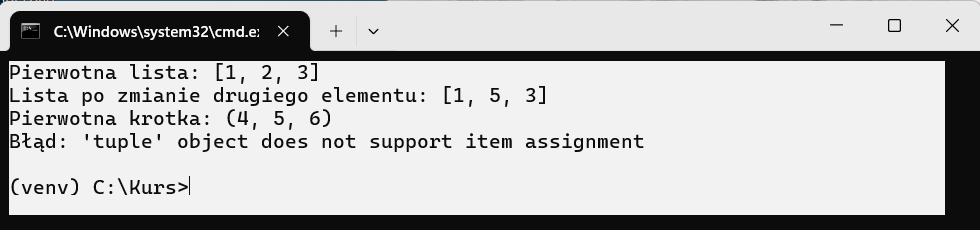
**Zadanie 1.6.3.**

|  |  |
| --- | --- |
| |  | | --- | | # Zadanie: 1\_6\_3.py  x = **5**  y = **3.14**  wynik1 = x + y  print(f"Wynik operacji x + y: {wynik1}")  print(f"Typ wyniku: {type(wynik1)}")  wynik2 = str(x) + str(y)  print(f"Wynik operacji str(x) + str(y): {wynik2}")  print(f"Typ wyniku: {type(wynik2)}") | |



**Zadanie 1.6.4. \***

|  |  |
| --- | --- |
| |  | | --- | | # Zadanie: 1\_6\_4.py  lista = [**1**, **2**, **3**]  print(f"Pierwotna lista: {lista}")  lista[**1**] = **5**  print(f"Lista po zmianie drugiego elementu: {lista}")  krotka = (**4**, **5**, **6**)  print(f"Pierwotna krotka: {krotka}")  **try**:  krotka[**1**] = **7**  **except** **TypeError** **as** e:  print(f"Błąd: {e}") | |



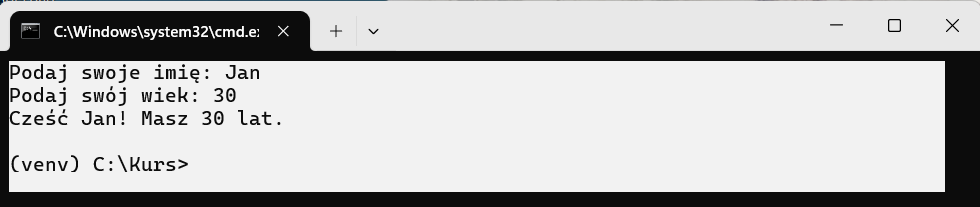
**Zadanie 1.6.5. \***

|  |  |
| --- | --- |
| |  | | --- | | # Zadanie: 1\_6\_5.py  produkt = {  "nazwa": "jabłko",  "cena": **3.5**,  "ilość": **10**  }  print(produkt) | |



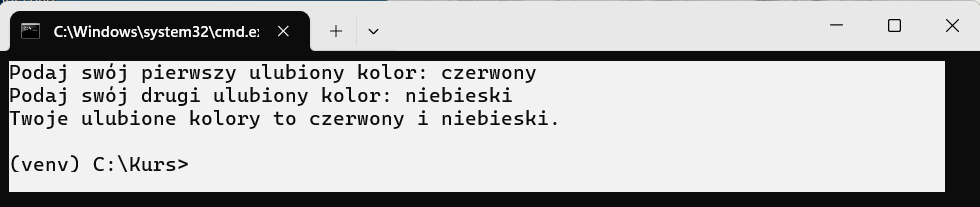
**Zadanie 1.7.1. Wczytaj imię i wiek**

|  |  |
| --- | --- |
| |  | | --- | | # Zadanie: 1\_7\_1.py  # Input: Jan,30  imie = input("Podaj swoje imię: ")  wiek = input("Podaj swój wiek: ")  print(f"Cześć {imie}! Masz {wiek} lat.") | |



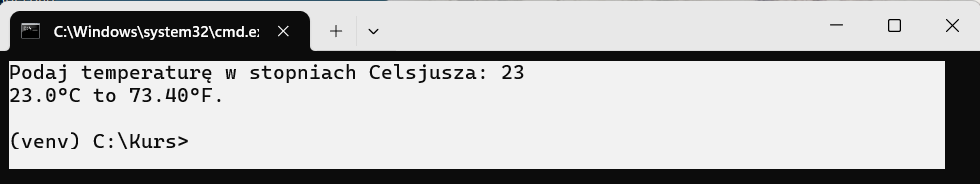
**Zadanie 1.7.2. Ulubione kolory**

|  |  |
| --- | --- |
| |  | | --- | | # Zadanie: 1\_7\_2.py  # Input: czerwony,niebieski  kolor1 = input("Podaj swój pierwszy ulubiony kolor: ")  kolor2 = input("Podaj swój drugi ulubiony kolor: ")  print(f"Twoje ulubione kolory to {kolor1} i {kolor2}.") | |



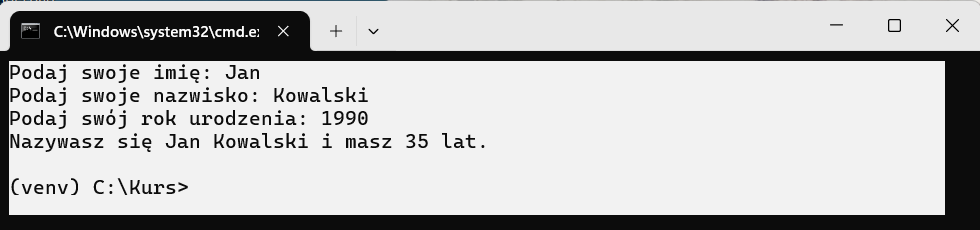
**Zadanie 1.7.3. Przeliczanie temperatury**

|  |  |
| --- | --- |
| |  | | --- | | # Zadanie: 1\_7\_3.py  # Input: 23  celsius = float(input("Podaj temperaturę w stopniach Celsjusza: "))  fahrenheit = (celsius \* **9** / **5**) + **32**  print(f"{celsius}°C to {fahrenheit:.2f}°F.") | |



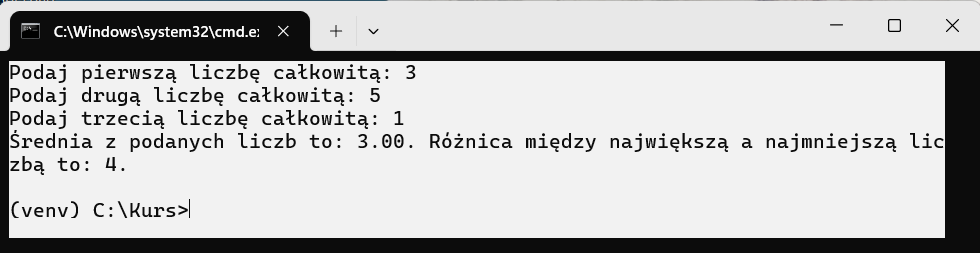
**Zadanie 1.7.4. \***

|  |  |
| --- | --- |
| |  | | --- | | # Zadanie: 1\_7\_4.py  # Input: Jan,Kowalski,1990  **from** **datetime** **import** datetime  imie = input("Podaj swoje imię: ")  nazwisko = input("Podaj swoje nazwisko: ")  rok\_urodzenia = int(input("Podaj swój rok urodzenia: "))  aktualny\_rok = datetime.now().year  wiek = aktualny\_rok - rok\_urodzenia  print(f"Nazywasz się {imie} {nazwisko} i masz {wiek} lat.") | |

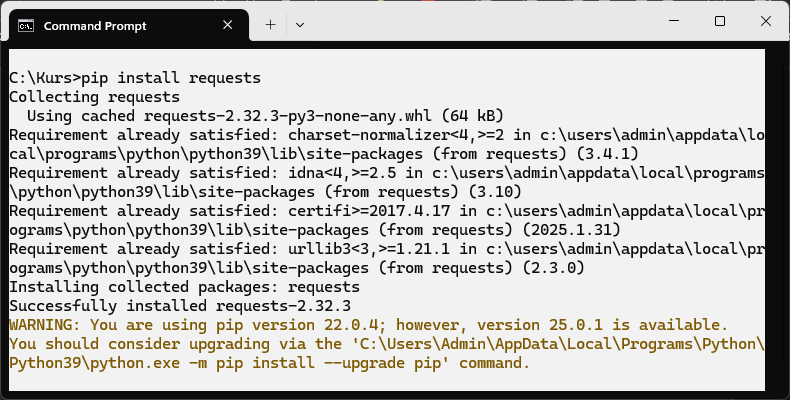


**Zadanie 1.7.5. \***

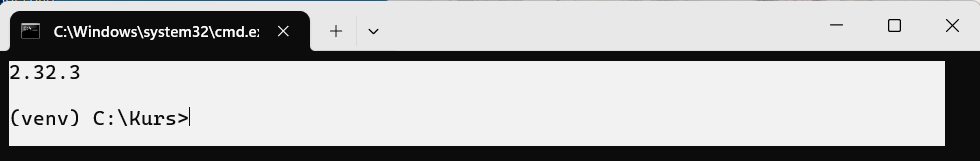
|  |  |
| --- | --- |
| |  | | --- | | # Zadanie: 1\_7\_5.py  # Input: 3,5,1  liczba1 = int(input("Podaj pierwszą liczbę całkowitą: "))  liczba2 = int(input("Podaj drugą liczbę całkowitą: "))  liczba3 = int(input("Podaj trzecią liczbę całkowitą: "))  srednia = (liczba1 + liczba2 + liczba3) / **3**  maksymalna = max(liczba1, liczba2, liczba3)  minimalna = min(liczba1, liczba2, liczba3)  roznica = maksymalna - minimalna  print(f"Średnia z podanych liczb to: {srednia:.2f}. Różnica między największą a najmniejszą liczbą to: {roznica}.") | |



**Zadanie 1.8.1.**

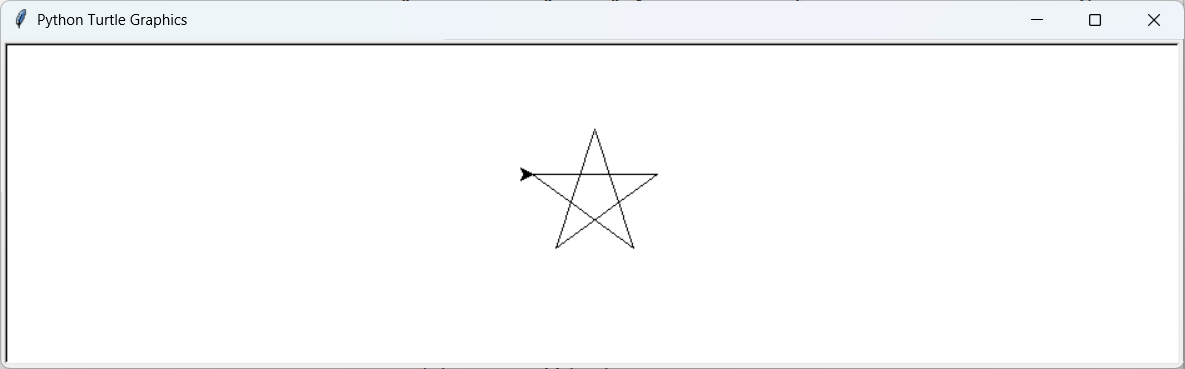


|  |  |
| --- | --- |
| |  | | --- | | # Zadanie: 1\_8\_1.py  **import** **requests**  print(requests.\_\_version\_\_) | |

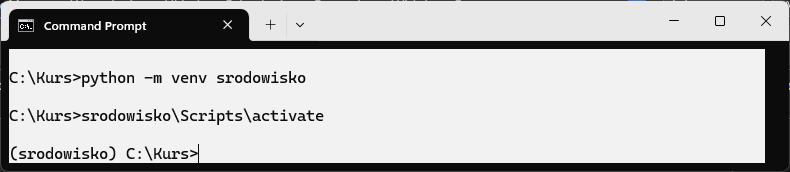


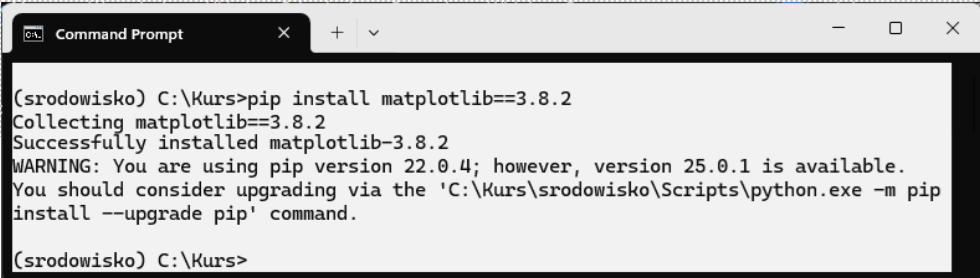
**Zadanie 1.8.2.**

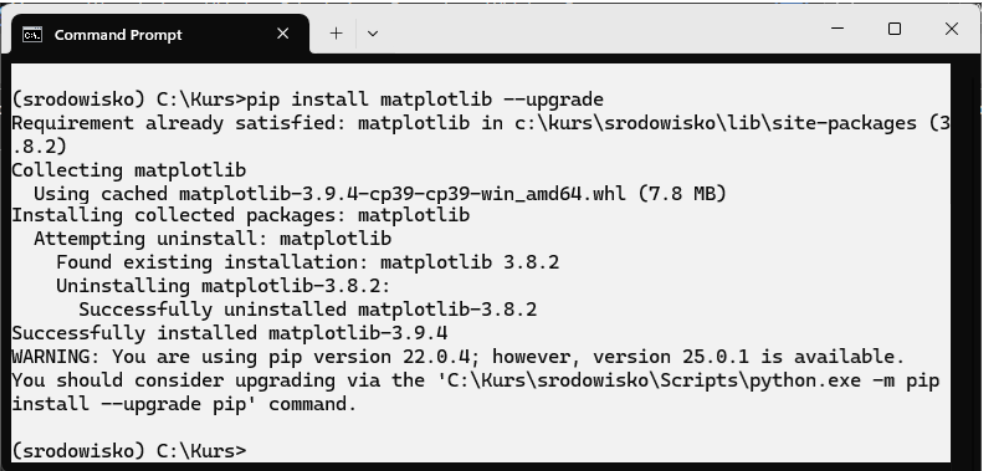
|  |  |
| --- | --- |
| |  | | --- | | # Zadanie: 1\_8\_2.py  **import** **turtle**  star = turtle.Turtle()  **for** \_ **in** range(**5**):  star.forward(**100**)  star.right(**144**)  turtle.done() | |



**Zadanie 1.8.3.**

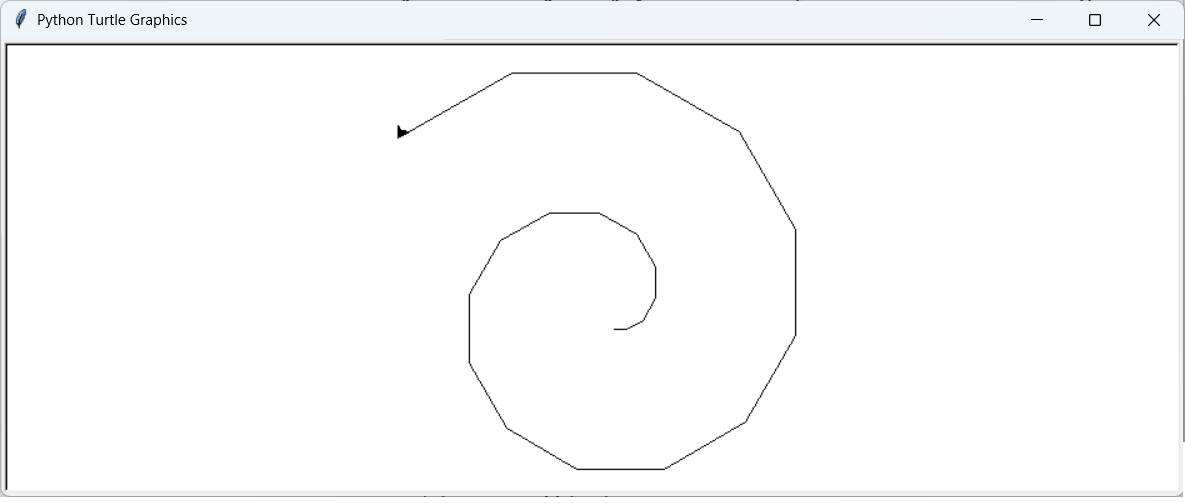






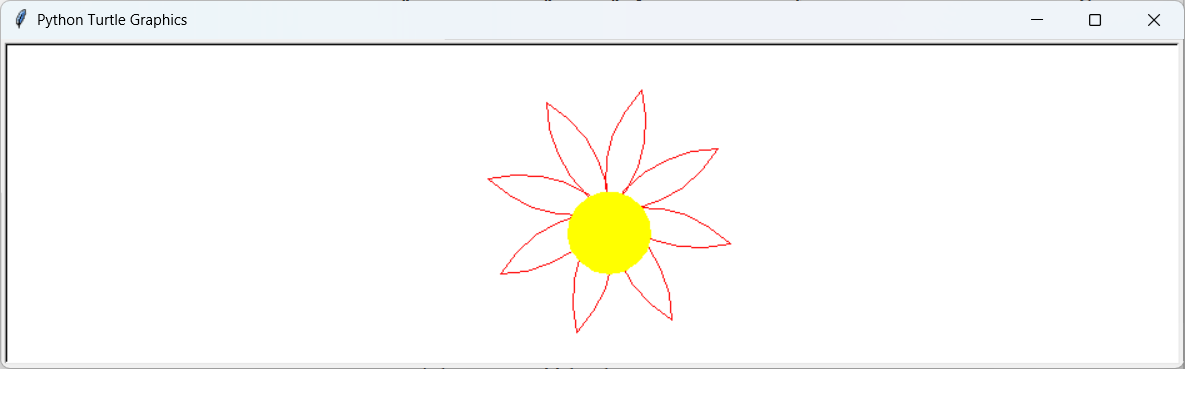
**Zadanie 1.8.4. \***

|  |  |
| --- | --- |
| |  | | --- | | # Zadanie: 1\_8\_4.py  **import** **turtle**  spiral = turtle.Turtle()  step\_length = **10**  **for** \_ **in** range(**20**):  spiral.forward(step\_length)  spiral.left(**30**)  step\_length += **5**  turtle.done() | |

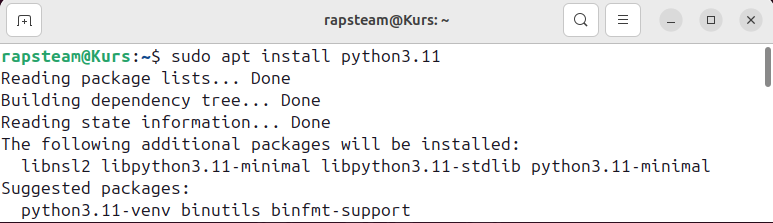


**Zadanie 1.8.5. \***

|  |  |
| --- | --- |
| |  | | --- | | # Zadanie: 1\_8\_5.py  **import** **turtle**  # Funkcja rysująca pojedynczy płatek  **def** **rysuj\_platek**(promien, kat):  **for** \_ **in** range(**2**): # Dwa łuki tworzą płatek  turtle.circle(promien, kat) # Rysuj łuk  turtle.left(**180** - kat) # Obrót żółwia, aby ustawić go do rysowania kolejnego łuku  # Funkcja rysująca kwiat  **def** **rysuj\_kwiat**(liczba\_platek, promien, kat\_platek):  kat\_obrotu = **360** / liczba\_platek # Kąt obrotu między płatkami  **for** \_ **in** range(liczba\_platek):  rysuj\_platek(promien, kat\_platek) # Rysuj płatek  turtle.left(kat\_obrotu) # Obrót żółwia do pozycji kolejnego płatka  # Rysowanie środka kwiatka  turtle.penup()  turtle.goto(**0**, -promien // **2**) # Przesuń żółwia na środek kwiatka  turtle.pendown()  turtle.color("yellow") # Środek kwiatka w kolorze żółtym  turtle.begin\_fill()  turtle.circle(promien // **3**) # Narysuj koło jako środek kwiatka  turtle.end\_fill()  turtle.speed("fastest") # Ustaw szybkość żółwia  turtle.color("red") # Kolor płatków  liczba\_platek = **8** # Liczba płatków  promien\_platek = **100** # Promień łuku płatka  kat\_platek = **60** # Kąt łuku płatka  rysuj\_kwiat(liczba\_platek, promien\_platek, kat\_platek)  turtle.hideturtle()  turtle.done() | |

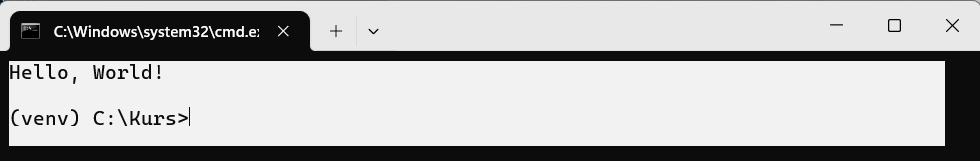


**Zadanie 1.10.1.**



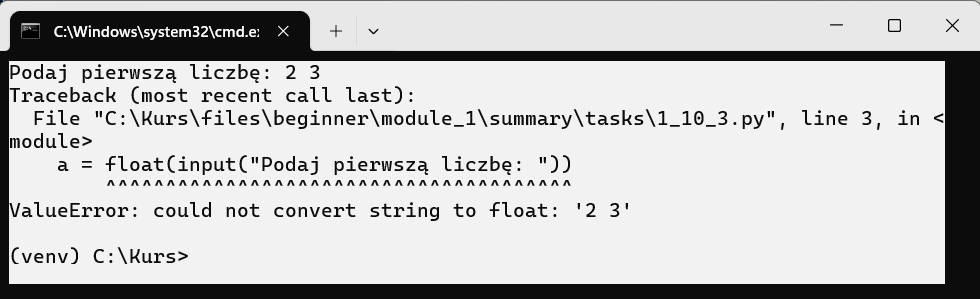
**Zadanie 1.10.2.**

|  |  |
| --- | --- |
| |  | | --- | | **if** \_\_name\_\_ == "\_\_main\_\_":  print("Hello, World!") | |



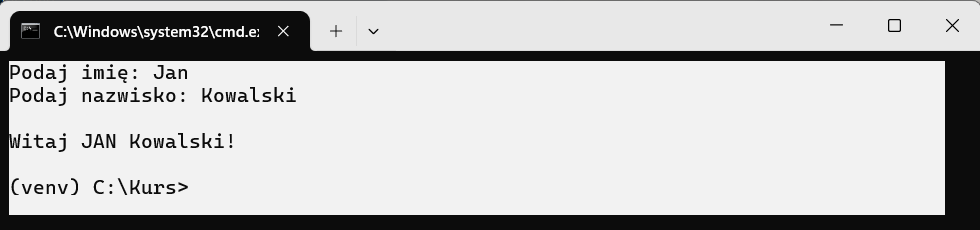
**Zadanie 1.10.3.**

|  |  |
| --- | --- |
| |  | | --- | | # Input: 2 3  a = float(input("Podaj pierwszą liczbę: "))  b = float(input("Podaj drugą liczbę: "))  suma = a + b  różnica = a - b  iloczyn = a \* b  iloraz = a / b  print(f"**\n**Suma: {suma}")  print(f"Różnica: {różnica}")  print(f"Iloczyn: {iloczyn}")  print(f"Iloraz: {iloraz}**\n**") | |



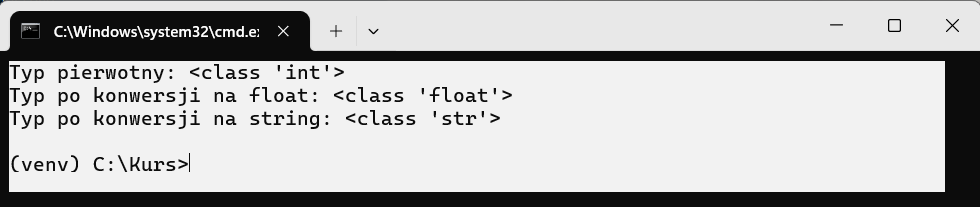
**Zadanie 1.10.4.**

|  |  |
| --- | --- |
| |  | | --- | | # Input: Jan,Kowalski  imie = input("Podaj imię: ").strip()  nazwisko = input("Podaj nazwisko: ").strip()  imie\_wielkie = imie.upper()  nazwisko\_cap = nazwisko.capitalize()  print(f"**\n**Witaj {imie\_wielkie} {nazwisko\_cap}!") | |



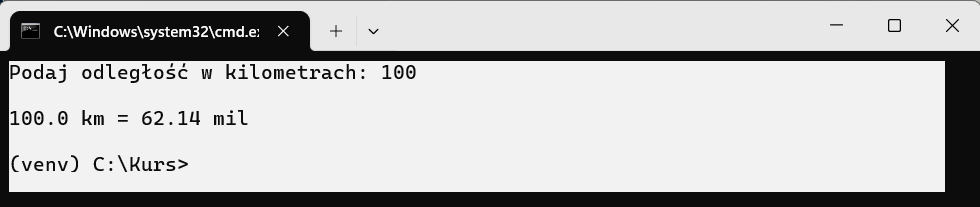
**Zadanie 1.10.5.**

|  |  |
| --- | --- |
| |  | | --- | | liczba = **21**  print(f"Typ pierwotny: {type(liczba)}")  liczba\_float = float(liczba)  print(f"Typ po konwersji na float: {type(liczba\_float)}")  liczba\_str = str(liczba)  print(f"Typ po konwersji na string: {type(liczba\_str)}") | |



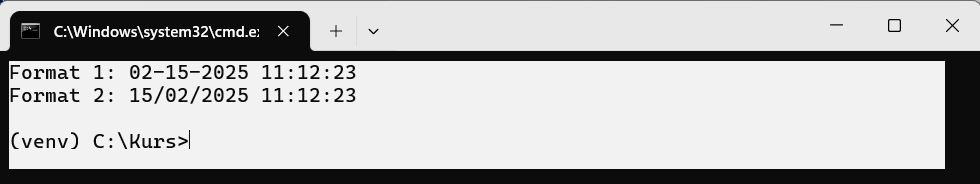
**Zadanie 1.10.6. \***

|  |  |
| --- | --- |
| |  | | --- | | # Input: 100  km = float(input("Podaj odległość w kilometrach: ").strip())  miles = km \* **0.621371**  print(f"**\n**{km} km = {miles:.2f} mil") | |



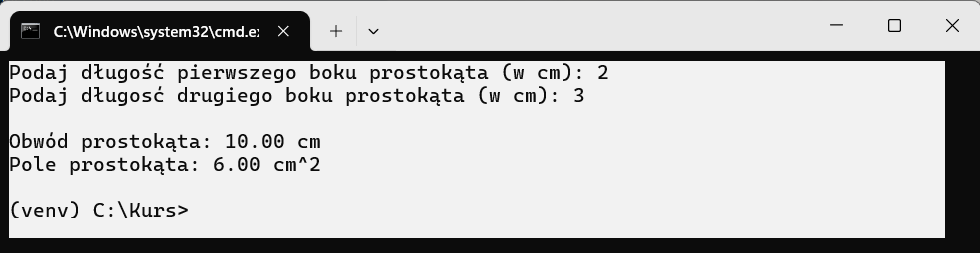
**Zadanie 1.10.7. \***

|  |  |
| --- | --- |
| |  | | --- | | **from** **datetime** **import** datetime  now = datetime.now()  format1 = now.strftime("%m-%d-%Y %H:%M:%S")  format2 = now.strftime("%d/%m/%Y %H:%M:%S")  print("Format 1:", format1)  print("Format 2:", format2) | |



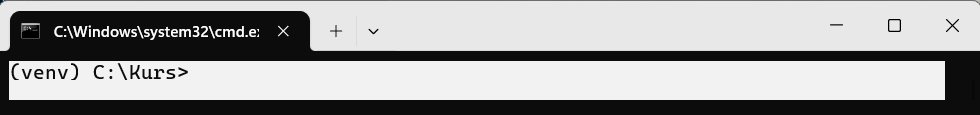
**Zadanie 1.10.8.**

|  |  |
| --- | --- |
| |  | | --- | | # Input: 2,3  a = float(input("Podaj długość pierwszego boku prostokąta (w cm): "))  b = float(input("Podaj długosć drugiego boku prostokąta (w cm): "))  obwod = **2** \* (a + b)  pole = a \* b  print(f"**\n**Obwód prostokąta: {obwod:.2f} cm")  print(f"Pole prostokąta: {pole:.2f} cm^2") | |



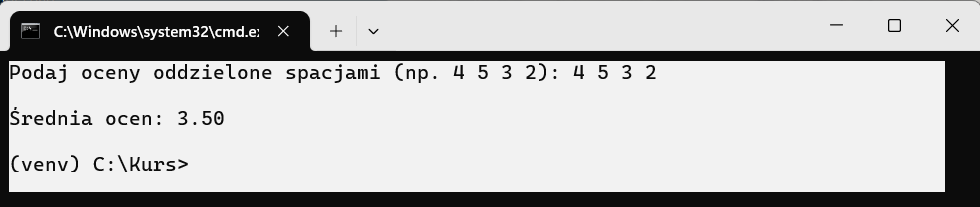
**Zadanie 1.10.9. \***

|  |  |
| --- | --- |
| |  | | --- | | # Input: 2,3  **import** **math**  promien = float(input("Podaj promień podstawy walca (w cm): "))  wysokosc = float(input("Podaj wysokość walca (w cm): "))  pole\_podstawy = math.pi \* promien \*\* **2**  objetosc = pole\_podstawy \* wysokosc  print(f"**\n**Pole podstawy: {pole\_podstawy:.2f} cm^2")  print(f"Objętość walca: {objetosc:.2f} cm^3") | |



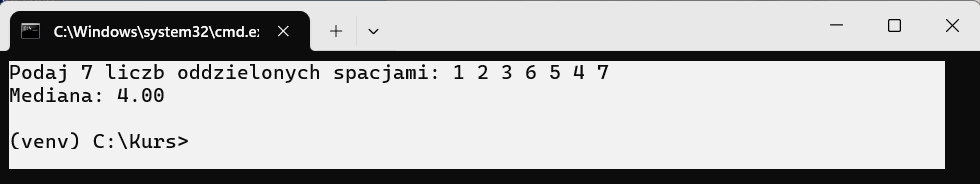
**Zadanie 1.10.10. \***

|  |  |
| --- | --- |
| |  | | --- | | # Input: 4 5 3 2  wprowadzenie = input("Podaj oceny oddzielone spacjami (np. 4 5 3 2): ").strip()  oceny = []  **for** ocena **in** wprowadzenie.split():  oceny.append(int(ocena))  srednia = sum(oceny) / len(oceny)  print(f"**\n**Średnia ocen: {srednia:.2f}") | |



**Zadanie 1.10.11. \***

|  |  |
| --- | --- |
| |  | | --- | | # Input: 1 2 3 6 5 4 7  wprowadzenie = input("Podaj 7 liczb oddzielonych spacjami: ").strip()  liczby\_str = wprowadzenie.split()  liczby = []  **for** liczba **in** liczby\_str:  liczby.append(int(liczba))  zbior = set(liczby)  posortowane = list(zbior)  mediana = posortowane[**3**]  print(f"Mediana: {mediana:.2f}") | |



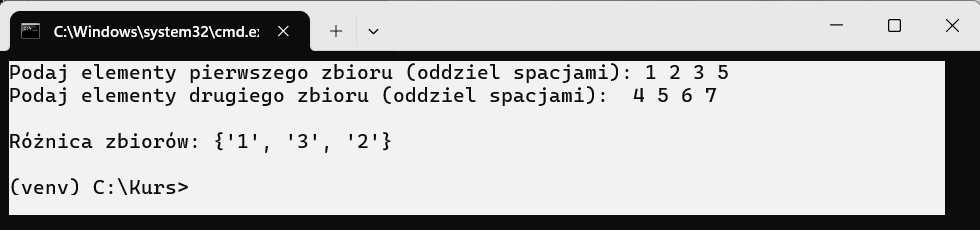
**Zadanie 1.10.12.**

|  |  |
| --- | --- |
| |  | | --- | | # Input: 1 2 3 5, 4 5 6 7  wprowadzenie1 = input("Podaj elementy pierwszego zbioru (oddziel spacjami): ").strip()  zbior1 = set(wprowadzenie1.split())  wprowadzenie2 = input("Podaj elementy drugiego zbioru (oddziel spacjami): ").strip()  zbior2 = set(wprowadzenie2.split())  suma\_zbiorow = zbior1 | zbior2  print("**\n**Suma zbiorów:", suma\_zbiorow) | |



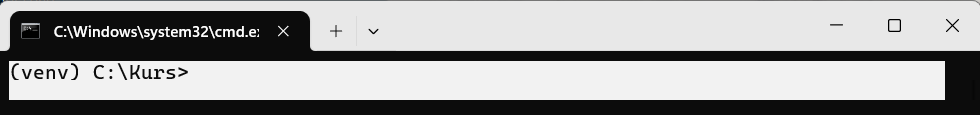
**Zadanie 1.10.13**

|  |  |
| --- | --- |
| |  | | --- | | # Input: 1 2 3 5, 4 5 6 7  wprowadzenie1 = input("Podaj elementy pierwszego zbioru (oddziel spacjami): ").strip()  zbior1 = set(wprowadzenie1.split())  wprowadzenie2 = input("Podaj elementy drugiego zbioru (oddziel spacjami): ").strip()  zbior2 = set(wprowadzenie2.split())  roznica\_zbiorow = zbior1 - zbior2  print("**\n**Różnica zbiorów:", roznica\_zbiorow) | |



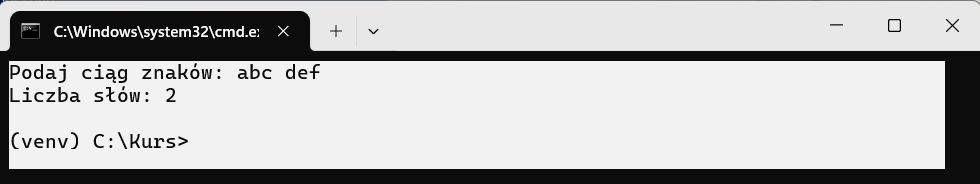
**Zadanie 1.10.14**

|  |  |
| --- | --- |
| |  | | --- | | # Input: 1 2 3 5, 4 5 6 7  wprowadzenie1 = input("Podaj elementy pierwszego zbioru (oddziel spacjami): ").strip()  zbior1 = set(wprowadzenie1.split())  wprowadzenie2 = input("Podaj elementy drugiego zbioru (oddziel spacjami): ").strip()  zbior2 = set(wprowadzenie2.split())  suma\_roz\_zbiorow = zbior1 ^ zbior2  print("**\n**Suma rozłączna (XOR) zbiorów:", suma\_roz\_zbiorow) | |



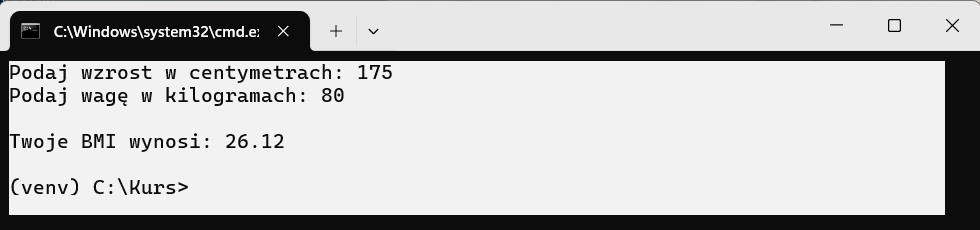
**Zadanie 1.10.15**

|  |  |
| --- | --- |
| |  | | --- | | # Input: abc def  tekst = input("Podaj ciąg znaków: ").strip()  lista\_slow = tekst.split()  print("Liczba słów:", len(lista\_slow)) | |



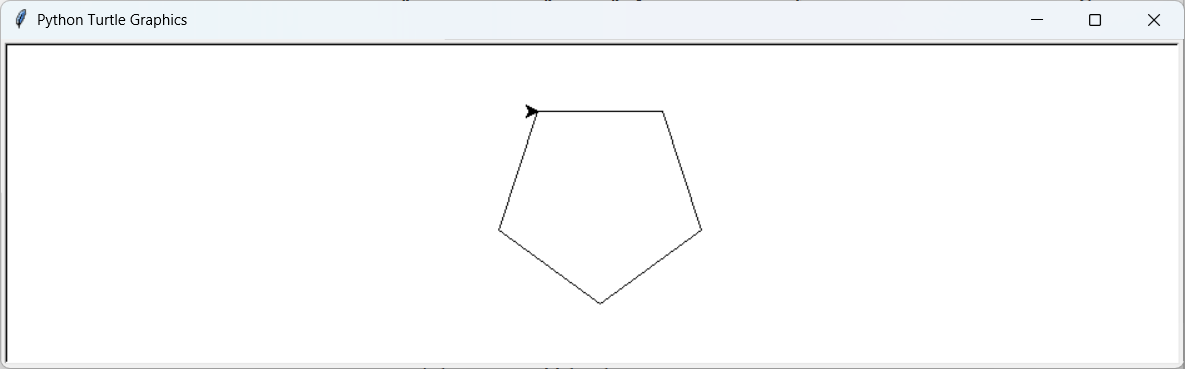
**Zadanie 1.10.16**

|  |  |
| --- | --- |
| |  | | --- | | # Input: 175,80  wzrost\_cm = float(input("Podaj wzrost w centymetrach: "))  waga\_kg = float(input("Podaj wagę w kilogramach: "))  wzrost\_m = wzrost\_cm / **100**  bmi = waga\_kg / (wzrost\_m \*\* **2**)  print(f"**\n**Twoje BMI wynosi: {bmi:.2f}") | |



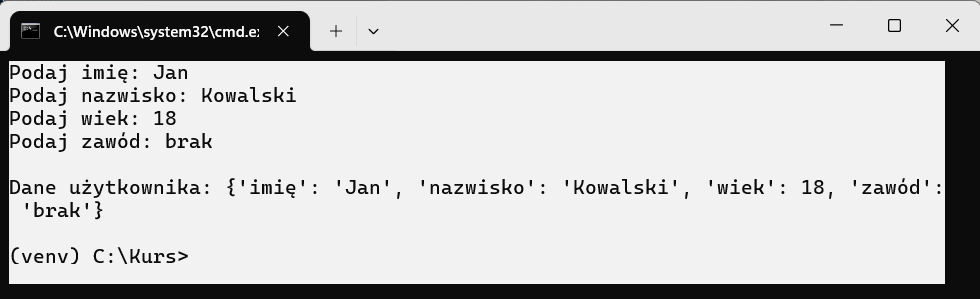
**Zadanie 1.10.17**

|  |  |
| --- | --- |
| |  | | --- | | # Input: 5  **import** **turtle**  liczba\_bokow = int(input("Podaj liczbę boków wielokąta: "))  t = turtle.Turtle()  kat = **360** / liczba\_bokow  **for** \_ **in** range(liczba\_bokow):  t.forward(**100**)  t.right(kat)  turtle.done() | |



**Zadanie 1.10.18**

|  |  |
| --- | --- |
| |  | | --- | | # Input: Jan,Kowalski,18,brak  imie = input("Podaj imię: ").strip()  nazwisko = input("Podaj nazwisko: ").strip()  wiek = int(input("Podaj wiek: ").strip())  zawod = input("Podaj zawód: ").strip()  dane\_uzytkownika = {  "imię": imie,  "nazwisko": nazwisko,  "wiek": wiek,  "zawód": zawod  }  print("**\n**Dane użytkownika:", dane\_uzytkownika) | |



**Zadanie 1.10.19**

|  |  |
| --- | --- |
| |  | | --- | | # Input: 5  liczba = int(input("Podaj liczbę od 1 do 10: "))  print(f"{liczba} x 1 = {liczba \* **1**}")  print(f"{liczba} x 2 = {liczba \* **2**}")  print(f"{liczba} x 3 = {liczba \* **3**}")  print(f"{liczba} x 4 = {liczba \* **4**}")  print(f"{liczba} x 5 = {liczba \* **5**}")  print(f"{liczba} x 6 = {liczba \* **6**}")  print(f"{liczba} x 7 = {liczba \* **7**}")  print(f"{liczba} x 8 = {liczba \* **8**}")  print(f"{liczba} x 9 = {liczba \* **9**}")  print(f"{liczba} x 10 = {liczba \* **10**}") | |



**Zadanie 1.10.20**

|  |  |
| --- | --- |
| |  | | --- | | # Input: 35  **import** **math**  **import** **turtle**  promien = float(input("Podaj promień okręgu: "))  t = turtle.Turtle()  t.circle(promien)  obwod = **2** \* math.pi \* promien  pole = math.pi \* promien \*\* **2**  print(f"Obwód okręgu: {obwod:.2f}")  print(f"Pole okręgu: {pole:.2f}")  turtle.done() | |

# Rozpoczynamy kodowanie w języku Python

## Przygotowanie środowiska IDE

### Instalacja Thony i Visual Studio Code w systemie Linux

### Instalacja Thony i Visual Studio Code w systemie Windows

Wytyczne

1. Po zaakceptowaniu Szczegółowego programu szkolenia wraz z rozkładem materiału nauczania dla poziomu podstawowego oraz średniozaawansowanego, **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. Materiały szkoleniowe muszą być **wytworzone oddzielnie dla każdego poziomu szkolenia (poziom podstawowy i poziom średniozaawansowany**). **Materiał szkoleniowy dla danego rodzaju szkolenia powinien 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. Wszystkie materiały szkoleniowe muszą być wytworzone w języku polskim.
4. Objętość e-skryptu powinna odpowiadać treści szkolenia (przy czym minimalna liczba stron to 150 dla każdego poziomu).
5. **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.
6. Każdy plik w formacie tekstowym musi być przygotowany z wykorzystaniem standardowego formatu stron – A4 (210mm x 297mm) i marginesami: lewy margines – 2,5cm, prawy margines – 2,5cm, górny margines – 2,5cm, dolny margines – 2,5cm. Dolna stopka dokumentu musi posiadać numer strony znajdujący się po prawej stronie dokumentu, na każdej stronie za wyjątkiem strony pierwszej. Nagłówek musi posiadać miejsce na logotypy unijne a dolna stopka musi posiadać miejsce na dodanie logotypów realizatorów projektu. Każdy plik w formacie tekstowym opracowany przez Wykonawcę musi spełniać następujące kryteria: czcionka: Times New Roman, rozmiar: 12, interlinia: 1,2 wiersza, odstęp po: 0 pkt, Kolor: czarny Tło: brak. Materiał szkoleniowy w pliku formatu tekstowego musi posiadać minimum 2 stopnie nagłówków (Rozdział Główny – Nagłówek 1, Podrozdział pierwszego poziomu – Nagłówek 2), oraz maksymalnie 4 stopnie nagłówków. Rozdziały muszą rozpoczynać się od cyfry np.: 1. Instalacja oprogramowania. Rozmiary czcionek dla poszczególnych nagłówków są następujące: Nagłówek 1 – rozmiar 20, Nagłówek 2 – rozmiar 18, Nagłówek 3 – rozmiar 16, Nagłówek 4 – rozmiar 14..
   1. *Materiały szkoleniowe* tekstowe muszą zawierać treści dotyczące wszystkich tematów z *rozkładu materiału nauczania* w tym zagadnień objętych *Szczegółowym programem szkolenia* wraz z przykładami ich zastosowania. **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. *Trudne aspekty* danej części merytorycznej powinny zostać odpowiednio oznaczone. Preferowany styl oznaczenia: tekst umieszczony w ramce z odpowiednim oznaczeniem graficznym.
   3. *Trudne aspekty* mają jasno wskazać uczestnikowi/uczestniczce potencjalny problem oraz jego rozwiązanie.
   4. **Opis każdego tematu musi kończyć się co najmniej 5 *zadaniami praktycznymi* dla uczestnika/uczestniczki**. *Zadania praktyczne* muszą dotyczyć opisu tematu w materiale. **W zadaniach tych musi wystąpić 1 lub 2 *zadania praktyczne* o zwiększonym stopniu trudności, które powinny być oznaczone znakiem gwiazdki (\*) jako bardziej wymagające**.
   5. *Zadania praktyczne* muszą być inne niż zadania w przykładach i powinny służyć utrwalaniu wiedzy uczestnika/uczestniczki z danego materiału. *Zadania praktyczne* do danego tematu powinny wymagać od uczestnika/uczestniczki szkolenia wykorzystania wiedzy i umiejętności nabytych przy realizacji wcześniejszych tematów na szkoleniu. Ma to na celu wykorzystanie *zadań praktycznych* do utrwalania materiału przepracowanego wcześniej na szkoleniu.
   6. W tekstowych *materiałach szkoleniowych* dla uczestniczek/uczestników szkolenia na końcu każdego rozdziału materiału muszą znajdować się *zadania podsumowujące* uzyskaną wiedzę z całego rozdziału. **Minimalna ilość *zadań podsumowujących* to 20**.
   7. *Materiał szkoleniowy* musi zawierać elementy grywalizacji tj. zdobywanie doświadczenia, poziomów i odznak.
   8. **Na końcu każdego rozdziału tekstowego *materiału szkoleniowego* musi znajdować się podsumowanie**. Podsumowanie musi zawierać w zwięzłej postaci najważniejsze informacje, które uczestnik/uczestniczka szkolenia przepracował/a przy realizacji tematów rozdziału. Podsumowanie musi przedstawiać aspekt teoretyczny jak i praktyczny. Podsumowanie nie może przekraczać rozmiaru 3 stron wybranego formatu.
   9. **Każdy rozdział tekstowego *materiału szkoleniowego* musi zawierać *dodatek z rozwiązaniami zadań* w którym przedstawione są kompletne rozwiązania wszystkich zadań w rozdziale**.
   10. **Jeżeli dane zadanie jest typu programistycznego, to oczekiwany wynik działania skryptu/programu musi być przedstawiony w formie graficznej jako zrzut ekranu. Ponadto, jego rozwiązanie musi zawierać w szczególności informacje o strukturach danych, algorytmie oraz pełną treść kodu w danym języku programowania tak aby jego wykonanie na komputerze pozwoliło uzyskać oczekiwane wyniki co do ich treści jak i formatowania**.
   11. **W *dodatku z rozwiązaniami zadań* mogą znajdować się tylko i wyłącznie rozwiązania zadań**. Muszą one zostać odpowiednio oznaczone aby uczestnik/uczestniczka mógł/a łatwo zidentyfikować zadanie w przypadku chęci skorzystania z gotowego rozwiązania.

1. Strona projektu Python - https://www.python.org/ [↑](#footnote-ref-2)
2. PyPI · The Python Package Index - https://pypi.org/ [↑](#footnote-ref-3)