Nauka programowania w językach Scratch i Python

Python – poziom podstawowy

Materiał dla nauczycieli

Spis treści

[1. Podstawy języka Python 2](#_Toc1646308944)

[1.1. Język Python, jego rola i podstawowe narzędzia 3](#_Toc720878228)

[1.1.1. Wprowadzenie 3](#_Toc591744121)

[1.1.1.1. Struktura kursu i sposób nauki 3](#_Toc1782941949)

[1.1.1.2. Zastosowanie języka 3](#_Toc1179714858)

[1.1.1.3. Wady i zalety języka Python 4](#_Toc653125950)

[1.1.1.4. Wykorzystanie języka Python – interaktywnie czy przez skrypt? 4](#_Toc294452007)

[1.1.2. Algorytmika – czym jest? 5](#_Toc607247258)

[1.1.3. Podstawowe narzędzia kursu 6](#_Toc1533587440)

[1.1.3.1. Systemy operacyjne (Windows oraz Ubuntu) 6](#_Toc1634747211)

[1.1.3.2. Środowiska i edytory (Thonny, Visual Studio Code) 6](#_Toc2118561411)

[1.1.3.3. Uruchamianie kodu języka Python w terminalu 7](#_Toc1256922823)

[1.1.4. Zadania praktyczne 9](#_Toc2115383989)

[1.2. Instalacja środowiska Python w różnych systemach operacyjnych 9](#_Toc2144734171)

[1.2.1. Wyjaśnienie czym jest terminal 9](#_Toc97738428)

[1.2.2. Poznanie podstawowych komend do poruszaniu się wewnątrz terminala 9](#_Toc1727826285)

[1.2.3. Instalacja środowiska Python na systemie Linux 12](#_Toc1948584165)

[1.2.4. Instalacja środowiska Python na systemie Windows 17](#_Toc1596195764)

[1.2.5. Weryfikacja wersji interpretera języka Python 20](#_Toc351107231)

[1.2.6. Tworzenie wirtualnych środowisk 21](#_Toc188051490)

[1.2.7. Zarządzanie wersjami 23](#_Toc1442257879)

[1.2.8. Zadania 24](#_Toc872462417)

[1.3. Zasady tworzenia i struktury skryptów w języku Python 25](#_Toc448014938)

[1.3.1. Struktura plików .py 25](#_Toc398217027)

[1.3.2. Pierwsze skrypty w edytorze tekstowym 26](#_Toc74637786)

[1.3.3. Uruchamianie skryptów Python w konsoli systemowej 26](#_Toc128209624)

[1.3.4. Wybór trybu pracy: interaktywny czy skryptowy? 27](#_Toc674011343)

[1.3.5. Zadania podsumowujące temat (TYLKO SKRYPTY!) 29](#_Toc983504506)

[1.4. Uruchamianie kodu w konsoli za pomocą interpretera 30](#_Toc524248688)

[1.4.1. Praca z interaktywną konsolą Python 31](#_Toc609229307)

[1.4.2. Wprowadzanie pojedynczych komend i natychmiastowe obserwowanie wyników 32](#_Toc1282214192)

[1.4.3. Testowanie fragmentów kodu w trybie interaktywnym 32](#_Toc747513167)

[1.4.4. Zadania podsumowujące temat 34](#_Toc1542803878)

[1.5. Prowadzenie operacji matematycznych i manipulacji tekstowych 35](#_Toc1685482211)

[1.5.1. Podstawowe operacje arytmetyczne w języku Python 36](#_Toc1409655170)

[1.5.2. Manipulacja napisami 37](#_Toc1064585060)

[1.5.3. Wykorzystanie wbudowanych funkcji: len() i type() 39](#_Toc1251509005)

[1.5.4. Zadania podsumowujące temat 40](#_Toc2085035852)

[1.6. Poznanie typów zmiennych w Pythonie 40](#_Toc19025912)

[1.6.1. Typy liczbowe 42](#_Toc102984870)

[1.6.2. Wartość logiczna 43](#_Toc340593432)

[1.6.3. Typy sekwencyjne 44](#_Toc744231469)

[1.6.4. Typy zbiorów 46](#_Toc951485551)

[1.6.5. Typ braku wartości 47](#_Toc443735841)

[1.6.6. Zadania 47](#_Toc396215987)

[1.7. Używanie metod wejścia i wyjścia 48](#_Toc912758598)

[1.7.1. Funkcja input() 49](#_Toc1688361087)

[1.7.2. Funkcja print() 49](#_Toc1378235396)

[1.7.3. Zadania 51](#_Toc955944364)

[1.8. Korzystanie z bibliotek i menadżera PIP 51](#_Toc2020719131)

[1.8.1. Instalacja i wykorzystanie bibliotek z pomocą narzędzia PIP 51](#_Toc1627545721)

[1.8.2. Instalacja i korzystanie z bibliotek na przykładzie turtle 53](#_Toc592259590)

[1.9. Podsumowanie modułu 54](#_Toc870181146)

[1.10. Zadania podsumowujące moduł 54](#_Toc2033637862)

[2. Rozpoczynamy kodowanie w języku Python 55](#_Toc147755173)

[2.1. Przygotowanie środowiska IDE 56](#_Toc758952944)

[2.1.1. Instalacja Thony i Visual Studio Code w systemie Linux 56](#_Toc216276913)

[2.1.2. Instalacja Thony i Visual Studio Code w systemie Windows 56](#_Toc817024937)

Na końcu rozdziału (Nagłówek 1) obowiązkowe podsumowanie – wykorzystać styl Nagłówek 2.

Po nagłówku tekst wprowadzający, krótki 3-4 zdania

Podsumowanie – nagłówek 2

W ramach treści w module należy zapoznać się z punktami a, b, d, e, f, h.

Dodatki proponuję na samym końcu dokumentu

|  |  |
| --- | --- |
| Trudne aspekty | Tu należy wpisać trudne aspekty np. typowe błędy jakie mogą się pojawić i na co zwracać uwagę np. w pythonie nie stosuje się średnika tylko wcięcia, wielkość liter ma znaczenie itp.  Proszę sobie tą tabelkę całą kopiować |
| Tu będzie grafika wykrzyknika (todo) |

# 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.

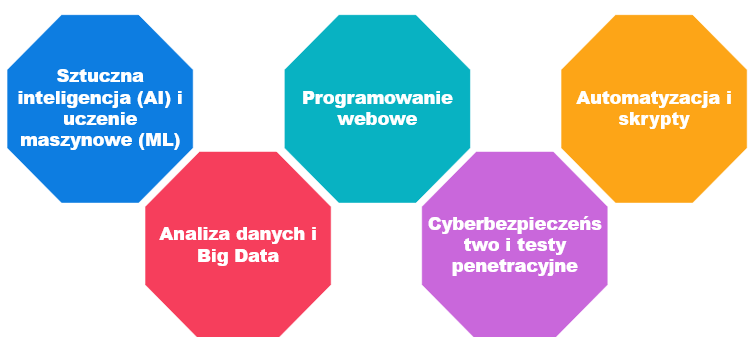
Sposób nauki opiera się na realizacji przykładów oraz wykonywaniu zadań utrwalających, co pozwala na aktywne przyswajanie materiału. Dla osób bardziej zaawansowanych przygotowano dodatkowe zadania (oznaczone \*) o wyższym stopniu trudności, które umożliwiają pogłębienie wiedzy i rozwój umiejętności programistycznych.

#### 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 .. 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 Windows przez *python* lub *python3* w cmd lub PowerShell.

Tabela .. 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 .. 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 Pythonie algorytmy można implementować przy użyciu struktur danych i instrukcji sterujących, np. sortowanie listy liczb można wykonać za pomocą algorytmu sortowania bąbelkowego lub wbudowanej funkcji.

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

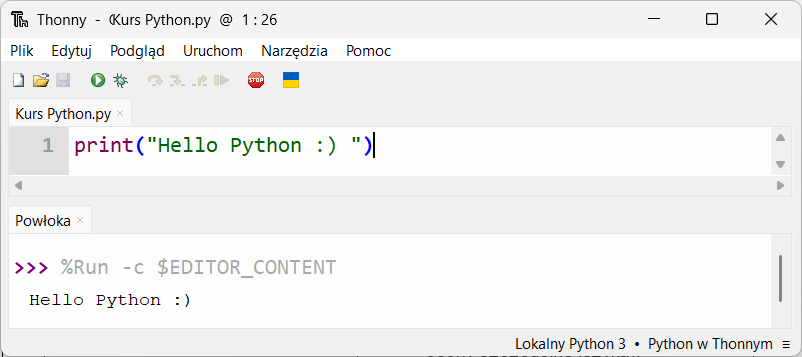
W trakcie kursu uczestnicy będą korzystać z różnych narzędzi wspomagających naukę i pracę z językiem Python. Obejmują one zarówno systemy operacyjne, na których działa Python, jak i środowiska programistyczne oraz edytory, które ułatwiają pisanie i uruchamianie kodu. W szczególności uczestnicy zapoznają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

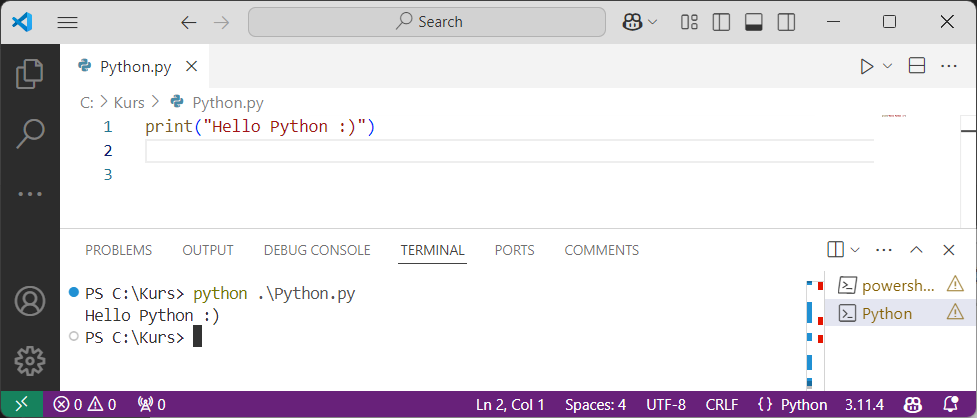
#### Środowiska i edytory (Thonny, Visual Studio Code)

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 1.2). 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 .. Thonny – prosty i darmowy edytor stworzony z myślą o nauce Pythona

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 1.3). 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 .. 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 *python nazwa\_pliku.py* lub *python3 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 .. Uruchamianie kodu języka Python w Wierszu Polecenia (CMD) w systemie 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 1.10.1. 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 1.10.2. Visual Studio Code**

* Uruchom Visual Studio Code oraz zmień motyw na jasny.

**Zadanie 1.10.3. Kolor konsoli systemowej**

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

**Zadanie 1.10.4. \* 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 1.10.5. \* Automatyczne formatowanie kodu**

* W Visual Studio Code włącz automatyczne formatowanie kodu.

## Instalacja środowiska Python w różnych systemach operacyjnych

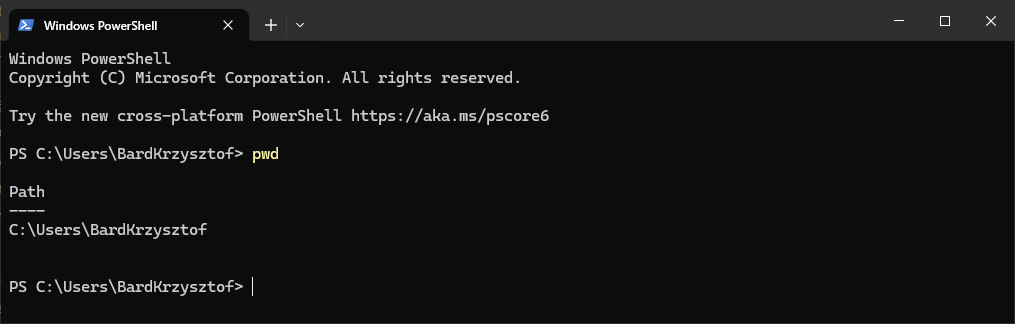
### Wyjaśnienie czym jest terminal

Terminal to narzędzie umożliwiające interakcję z systemem operacyjnym przy użyciu tekstowych poleceń, działające jako interfejs wiersza poleceń (CLI). Zamiast korzystać z graficznego interfejsu użytkownika (GUI), w terminalu użytkownik wpisuje komendy, a system odpowiada komunikatami tekstowymi, co pozwala na wykonywanie szerokiego zakresu operacji. Terminal oferuje użytkownikom bezpośredni dostęp do rdzenia systemu operacyjnego. Pozwala to na zarządzanie plikami, katalogami, uruchamianie programów, konfigurowanie ustawień systemowych oraz automatyzację zadań poprzez skrypty. W ten sposób, stanowi za interfejs pośredniczący między człowiekiem a komputerem. Terminal jest integralnym elementem systemów operacyjnych, zwłaszcza w środowiskach uniksowych (Linux, macOS) oraz w systemie Windows (np. Command Prompt, PowerShell).

### Poznanie podstawowych komend do poruszaniu się wewnątrz terminala

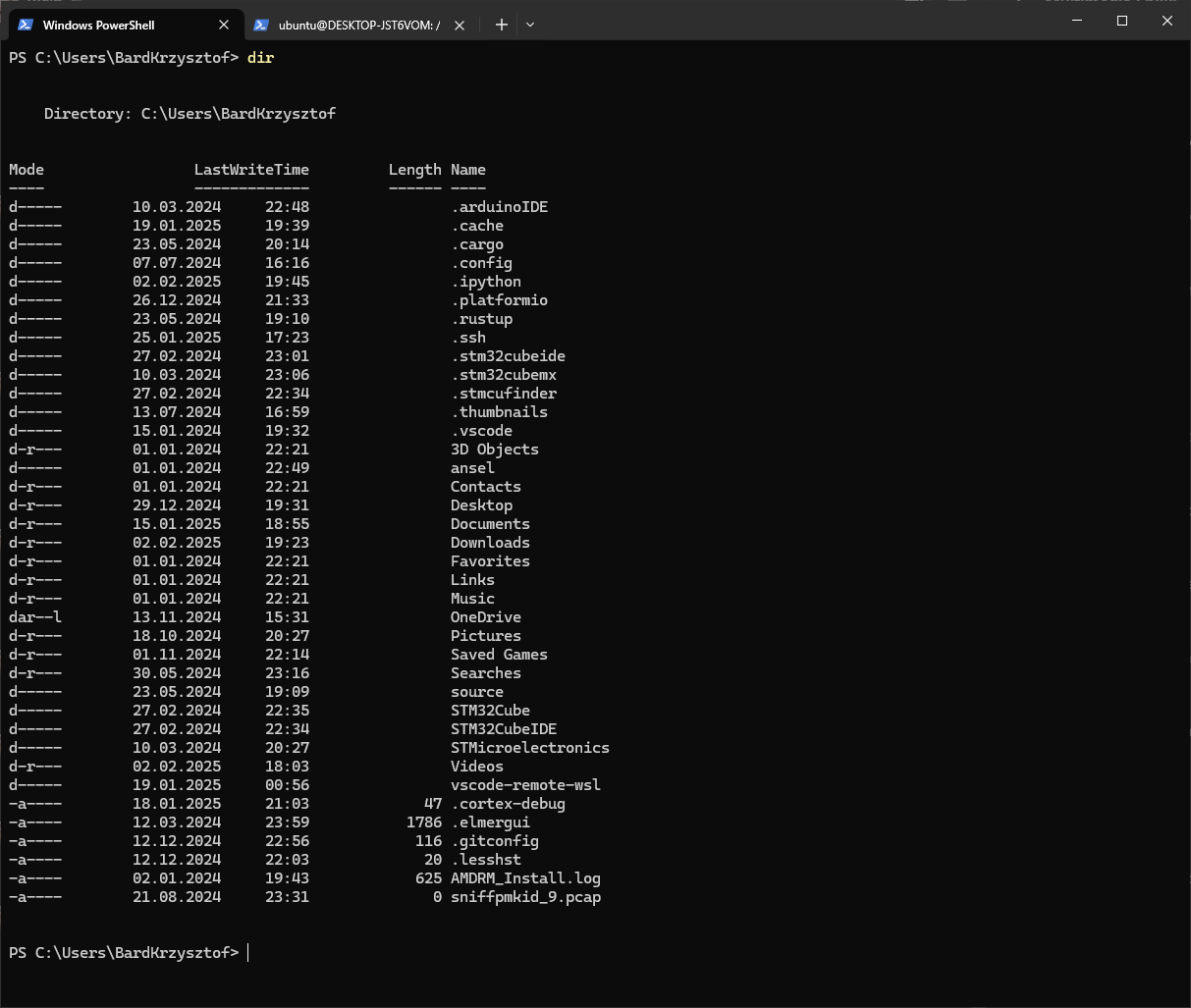
W terminalu polecenia są wprowadzane w formie tekstowej, a wyniki operacji wyświetlane są jako ciąg znaków. Aby móc się sprawnie po nim poruszać i realizować zadania, konieczne jest poznanie podstawowych komend.

* **pwd -** realizuje wypisanie ścieżki w jakiej aktualnie znajduje się użytkownik

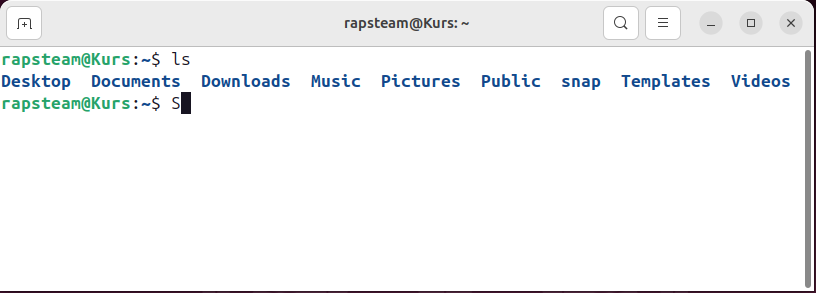


Rysunek .. Przykład użycia komendy pwd

* **ls** (linux) / **dir (**win) - wyświetlaZalety zawartość folderu, w którym aktualnie znajduje się użytkownik

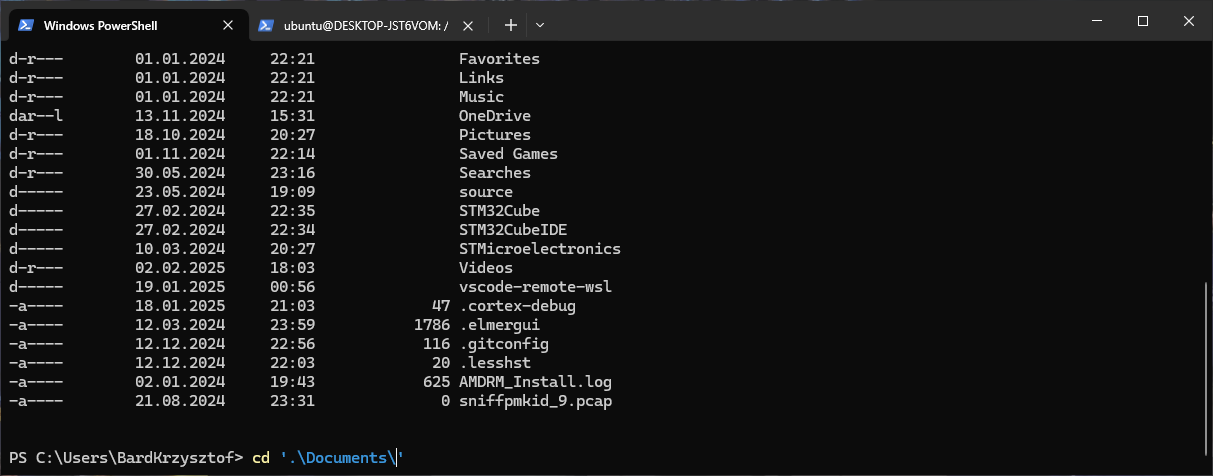


Rysunek .. Przykład użycia komendy dir (win)

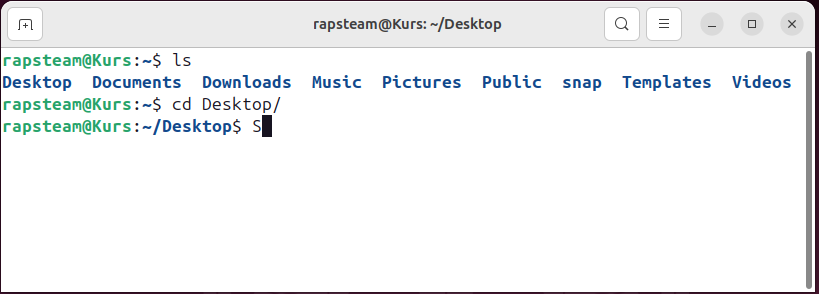


Rysunek .. Przykład użycia komendy ls (linux)

* **Cd –** zmiana katalogu w którym aktualnie ma się znaleźć użytkownik



Rysunek .. Przykład użycia komendy cd (win)



Rysunek .. Przykład użycia komendy cd (linux)

W terminalu szczególną uwagę może zwracać obecności kropki, czasamej występującej samodzielnie, a czasami przy nazwie plików. Służy ona przede wszystkim jako symbol reprezentujący bieżący katalog. W systemach operacyjnych opartych na Uniksie kropka umożliwia odwołanie się do aktualnej lokalizacji w strukturze katalogów, co pozwala na tworzenie relatywnych ścieżek dostępu do plików i katalogów. Polecenie uruchomienia skryptu z bieżącego katalogu często zapisuje się jako „./nazwa\_skryptu”, gdzie „./” oznacza „w bieżącym katalogu”. Zestawienie dwóch kropek („..”) odnosi się do katalogu nadrzędnego, czyli katalogu znajdującego się jeden poziom wyżej od bieżącego. Użycie tych symboli jest niezbędne podczas nawigacji w systemie plików.

* **.** - wskazanie na aktualny katalog
* **..** - wskazanie na katalog nadrzędny
* **.plik.txt** / **.folder**  - plik lub folder który jest "ukryty". Zazwyczaj przechowuje dane konfiguracyjne projektów lub aplikacji.

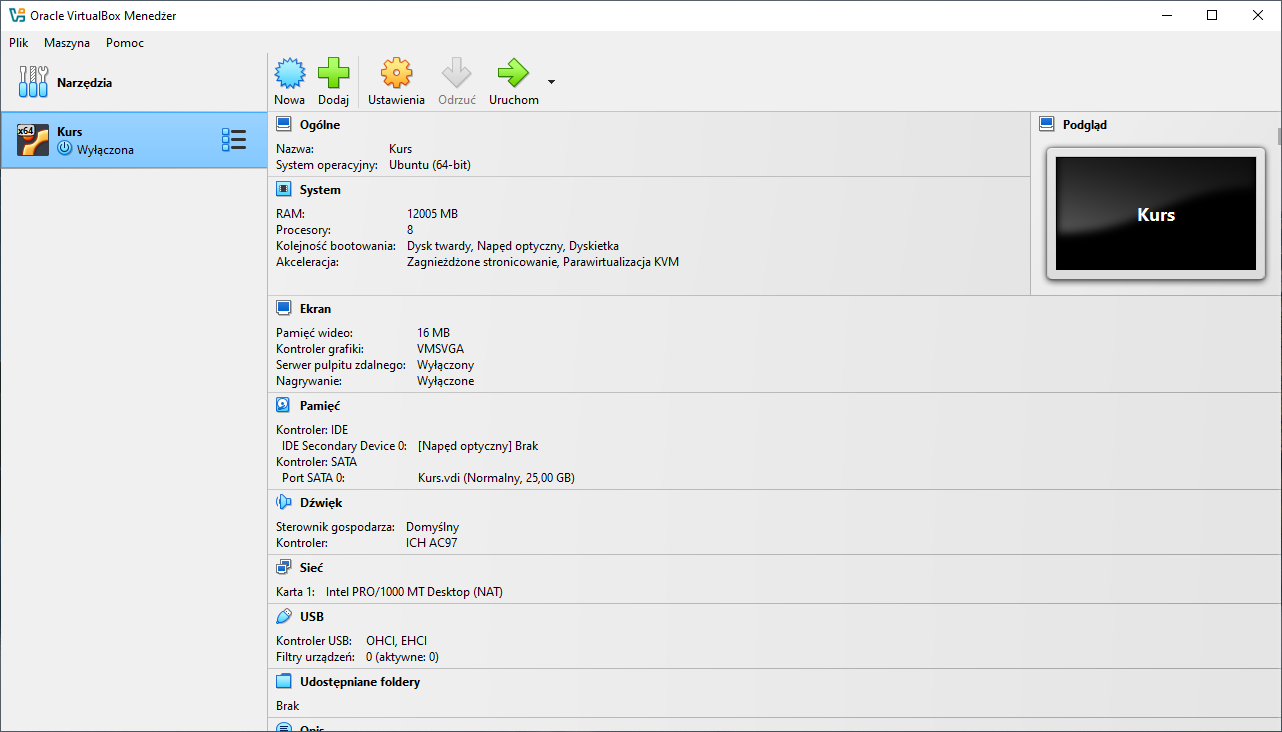
### Instalacja środowiska Python na systemie Linux

Przykład, jak zainstalować środowisko Python na systemie Linux, wymaga posiadania takowego, zainstalowanego natywnie na komputerze lub środowiska wirtualnego, emulującego system Linux. W poniższym przykładzie zostanie użyte oprogramowanie **Orale VirtualBox**, które zostało już skonfigurowane i posiada odpowiedni obraz. Najpierw należy uruchomić poniższą aplikację z poziomu pulpitu, lub wpisując jej nazwę w pasku wyszukiwania systemu.

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

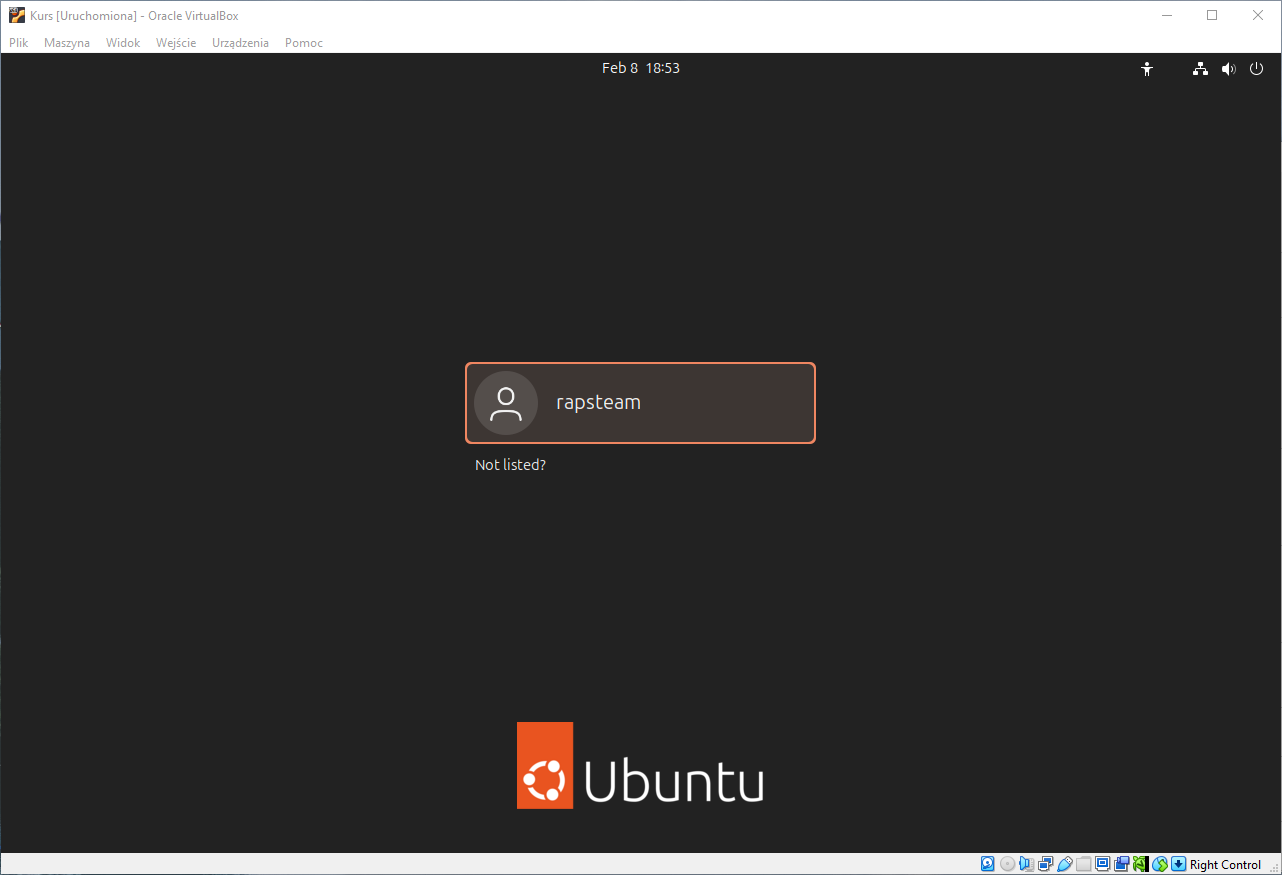
Rysunek . Ikona emulatora systemu Oracle VirtualBox (w zależności od wersji może się różnić od przedstawionej)

Po uruchomieniu, ukaże 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 środkowej części ekranu widać dane konfiguracyjne oraz przydzielone do niej zasoby. Gdyby była taka koniczność, można je modyfikować w menu „Ustawienia” widocznym nad informacjami.



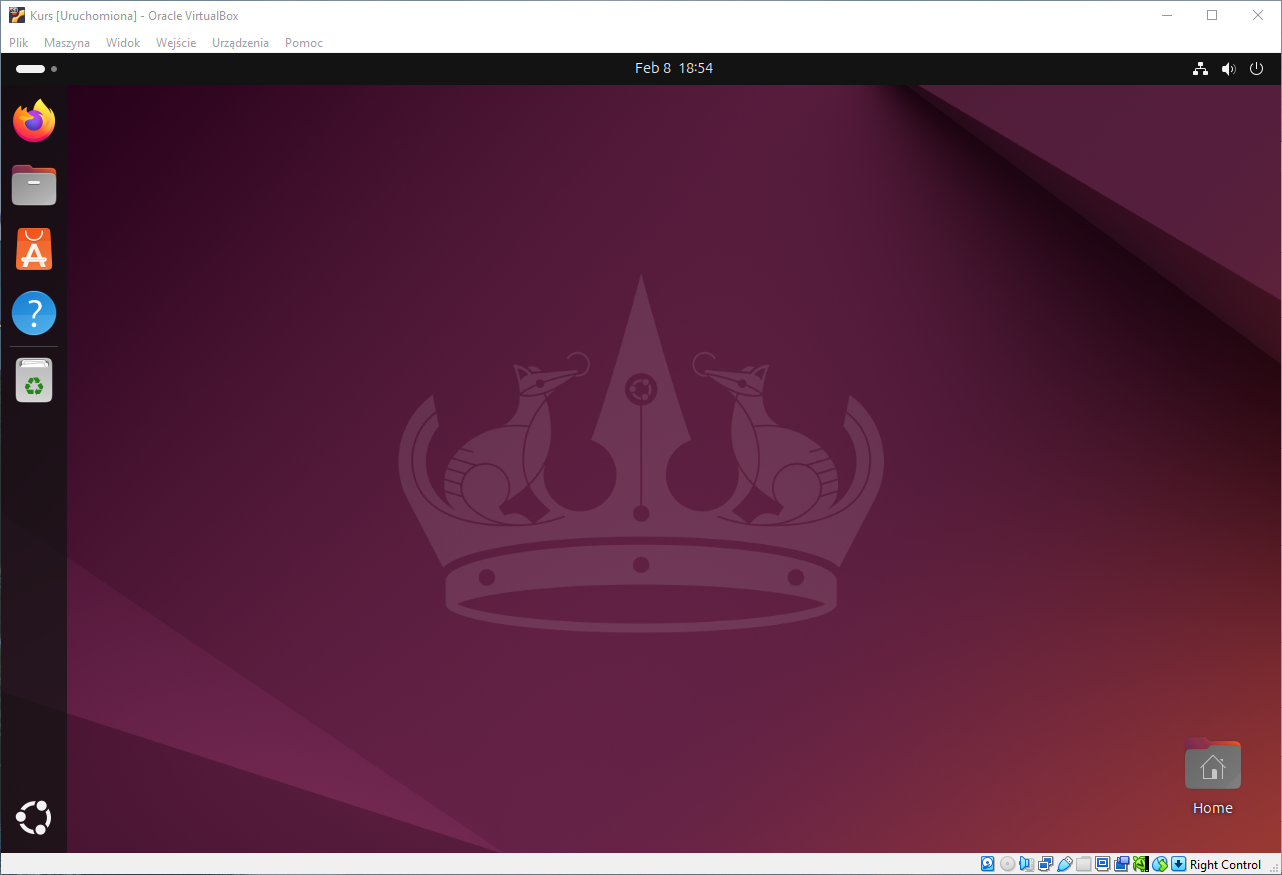
Rysunek . Okno wyboru skonfigurowanych maszyn wirtualnych

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 . 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.

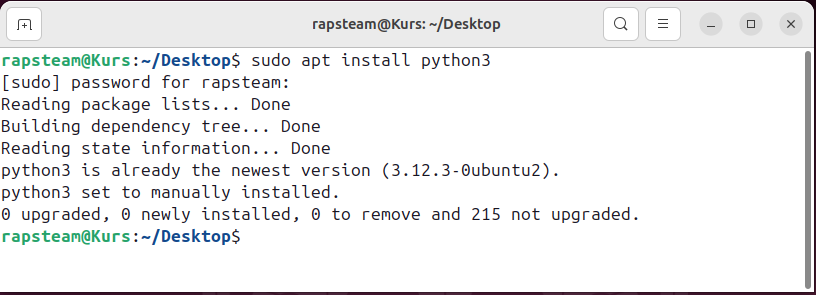


Rysunek . Pulpit systmeu Ubuntu

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.

Listing 1.1 Instalacja interpretera Python 3 w systemach opartych na Debianie

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

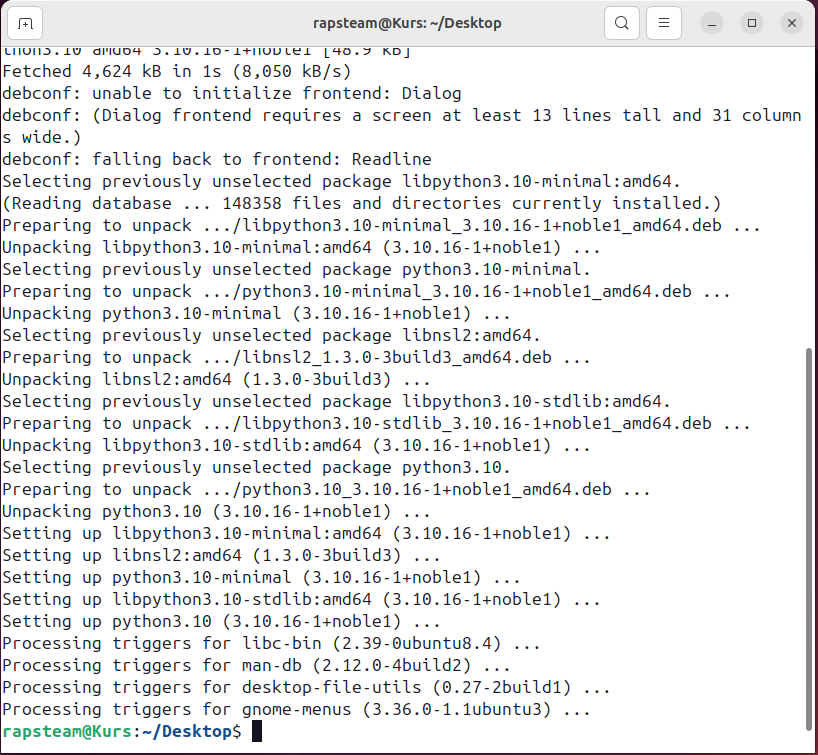


Rysunek . Efekt instalacji Pythona w wersji 3, w przypadku posiadania go już w systemie (Linux)

Możliwe jest także wskazanie konkretnej wersji pythona do instalacji. Może być to wykorzystywane w sytuacji posiadania wielu projektów korzystających z różnych wersji interpretera języka.

Listing . Instalacja interpretera Python 3 w wersji 3.10 w systemach opartych na Debianie

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



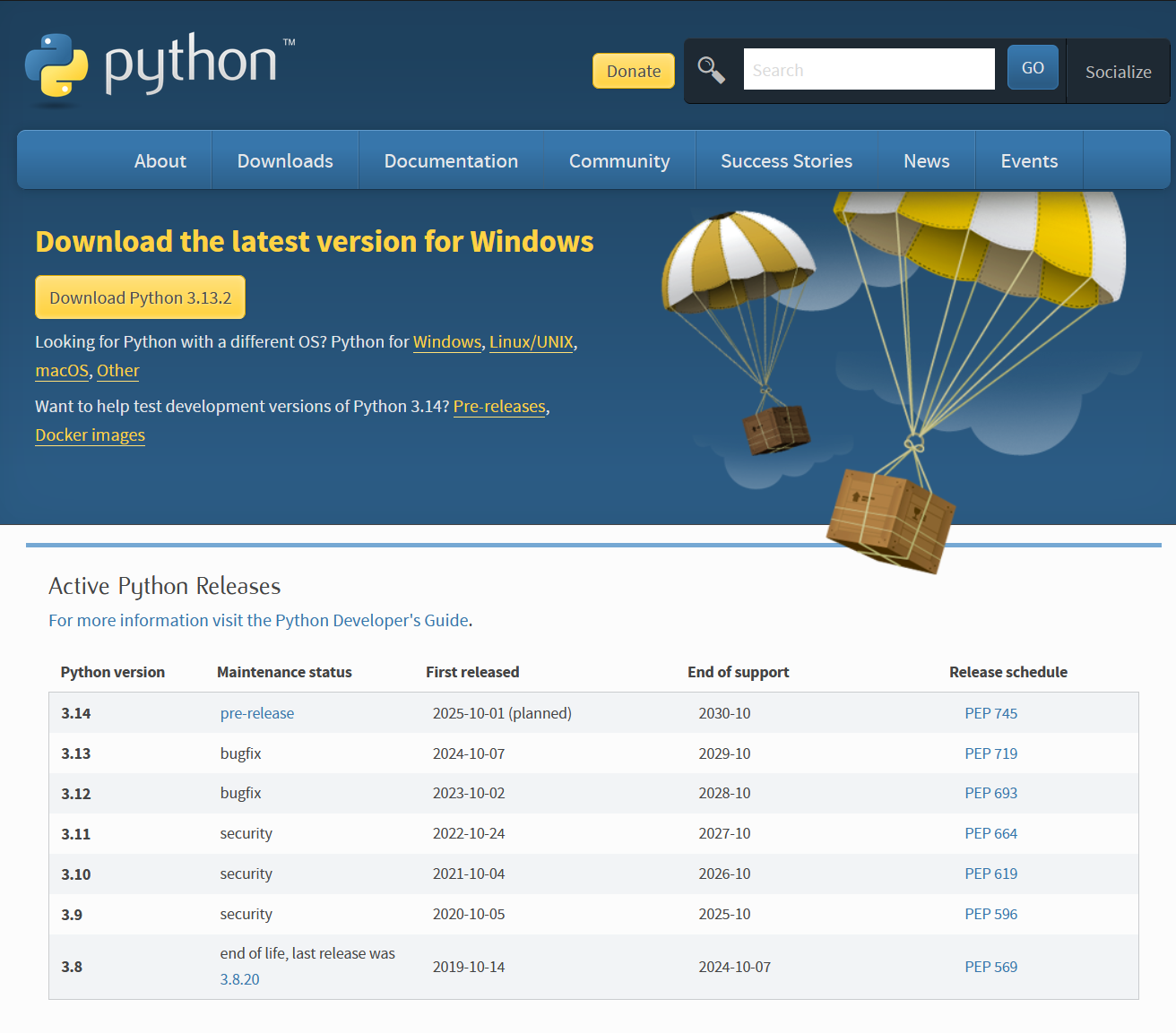
Rysunek . Efekt instalacji pythona w wersji szczególnej, w przypadku posiadania go w systemie (linux)

Podczas instalacji, konieczne było użycie polecenia *sudo* (ang. s*uper user do).* Umożliwia one wykonywanie poleceń z uprawnieniami administratora w systemach operacyjnych opartych na Uniksie. Pozwala to na tymczasowe podniesienie poziomu uprawnień, co jest wymagane przy wykonywaniu operacji wpływających na cały system, takich jak instalacja oprogramowania, modyfikacja plików systemowych czy zmiana ustawień zabezpieczeń. Użycie *sudo* wiąże się z koniecznością podania hasła użytkownika, co ogranicza możliwość przypadkowych lub nieautoryzowanych zmian w systemie. Po poleceniu *sudo* następuje wywołanie menadżera pakietów *apt* (Advanced Package Tool) z opcją *install.* Polecenie służy do instalacji oprogramowania w systemach opartych na dystrybucji Debiana, takich jak Ubuntu. Narzędzie zarządza pakietami oprogramowania, umożliwiając pobieranie ich z repozytoriów, instalację wraz z niezbędnymi zależnościami, a także aktualizację i usuwanie. Użycie apt install pozwala na łatwą i zautomatyzowaną instalację oprogramowania, zapewniając spójność i aktualność zainstalowanych pakietów. W innych dystrybucjach mogą występować inni menadżerzy tacy jak *dnf* (Fedora), *yum* (Red Hat) lub *pacman* (Arch).

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

### Instalacja środowiska Python na systemie Windows

Aby zainstalować środowisko Python na systemie Windows najpierw należy wejść na stronę <https://www.python.org/> i w zakładkę *Downloads*. Klikając ją pojawią się możliwe wersje interpretera python do pobrana. Jeżeli na systemie nie było wcześniej zainstalowanego środowiska python, można kliknąć w żółty przycisk *Download Python 3.13.1.* Wówczas rozpocznie się proces pobierania instalatora.



Rysunek . Oficjalna strona projektu Python, z której można pobrać interpreter

|  |  |
| --- | --- |
|  | Utrzymywane ciągłe wsparcie aktualizacjami dla języka Python, może powodować, że aktualnie najnowsza wersja instalatora jest inna niż ta wskazana na zrzucie ekranu. Dowiedz się jaka wersja jest aktualnie najnowsza lub pobierz taką, jaka będzie odpowiednia dla twojego projektu |

Po pobraniu pliku instalatora z rozszerzeniem *.exe* i uruchomieniu go, pokaże się ekran taki jak na rysunku 1.13 stanowiący ekran początkowy instalatora. Do dyspozycji są dwie opcje instalacji:

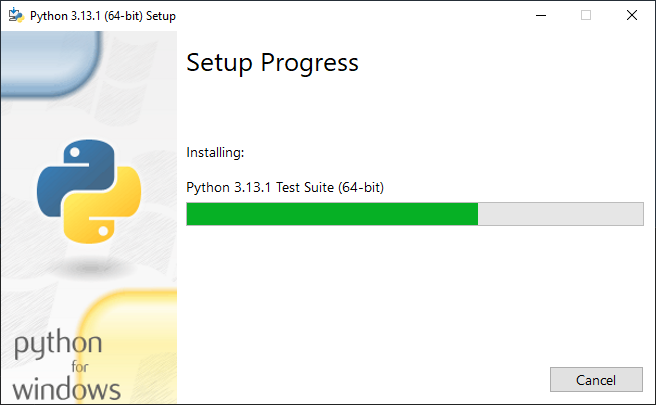
* Install Now (opcja zalecana) – instaluje interpreter w domyślnej, wskazanej przez program lokalizacji systemowej oraz załącza do tego procesu szereg podstawowych, najpopularniejszych bibliotek dla tego języka, menadżer pakietów PIP oraz ustawia skojarzenie plików z rozszerzeniem *.py* jako pliki uruchomieniowe przez interpreter języka python.
* Customize installation – spersonalizowana instalacja, w której użytkownik wybiera na jakiej ścieżce ma zostać zainstalowany interpreter oraz jakie funkcje i biblioteki mają być zawarte.

Przed wyborem jednej z tych dwóch opcji, **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,



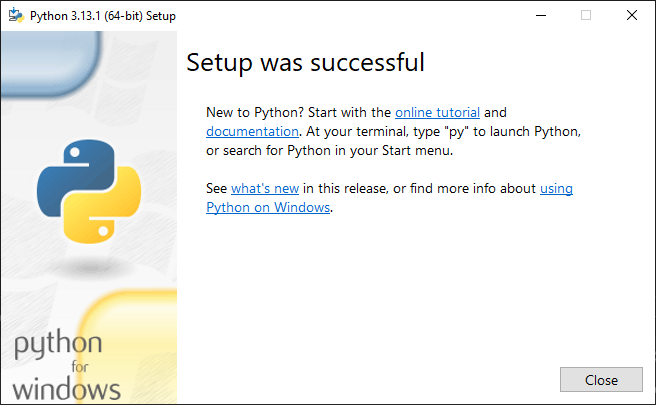
Rysunek . Okno instalatora interpretera Python (win)

Po kliknięciu *Install Now* pojawi się okno postępu instalacji takie, jak zaprezentowane na rysunku 1.14.



Rysunek . Okno postępu instalacji interpretera języka Python (win)

Po procesie instalacji pojawi się okno z informacją o jej pomyślnym zakończeniu.



Rysunek . Ekran końcowy instalatora Python 3.13.1 (64-bit) potwierdzający pomyślną instalację

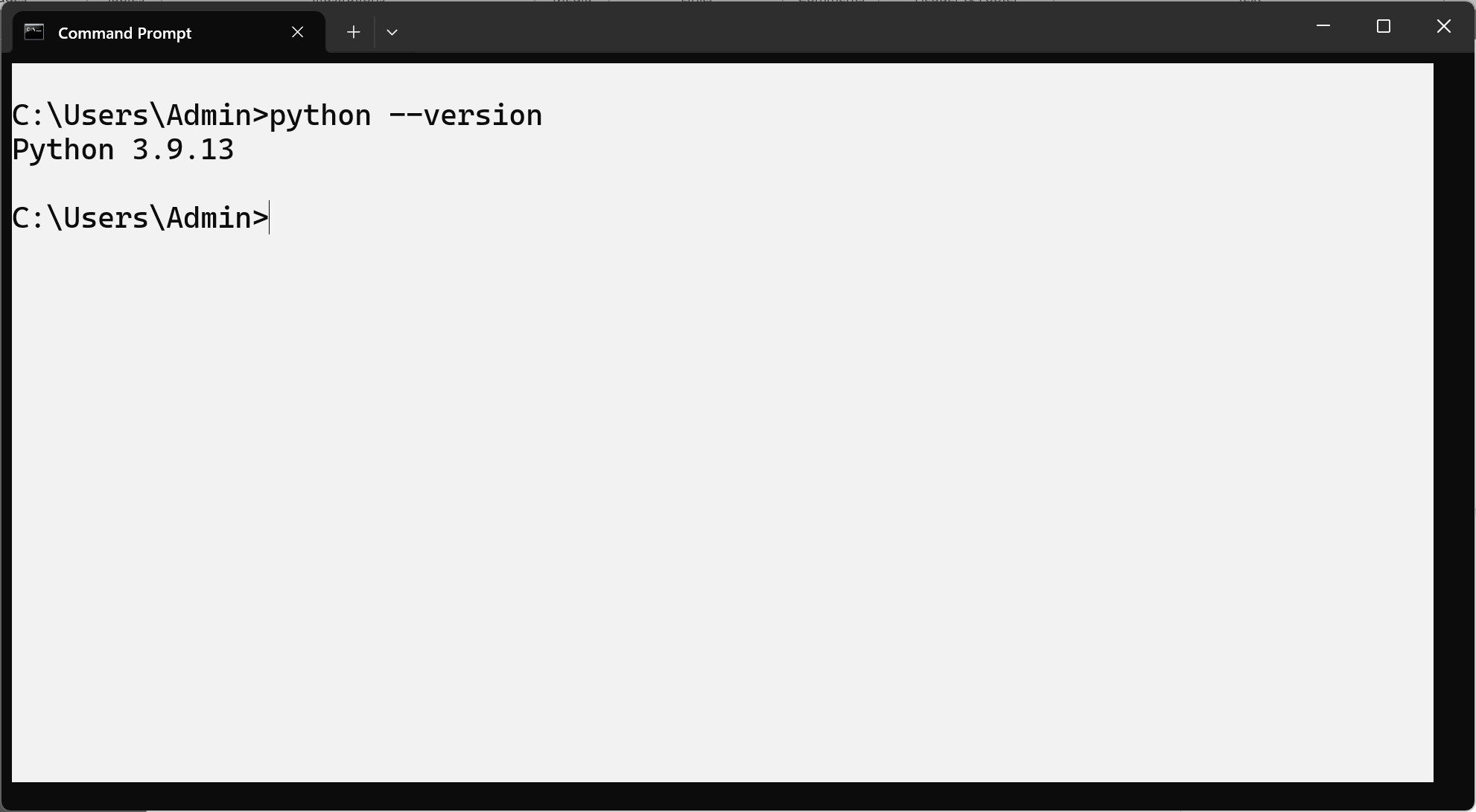
### Weryfikacja wersji interpretera języka Python

Gdy proces instalacji dojdzie do skutku, zarówno na systemie Windows oraz Linux można zweryfikować, czy wywołanie polecenia *python* spowoduje zadziałanie interpretera języka. Do tego celu posłuży terminal systemowy.

|  |  |
| --- | --- |
|  | W przypadku systemu Windows przed próbą wywołania polecenia, jeżeli po raz pierwszy instalowałeś na komputerze interpreter Pythona, konieczne jest jego ponowne uruchomienie. |

Listing . Komenda która wyświetla obecnie zainstalowaną wersję interpretera Python

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



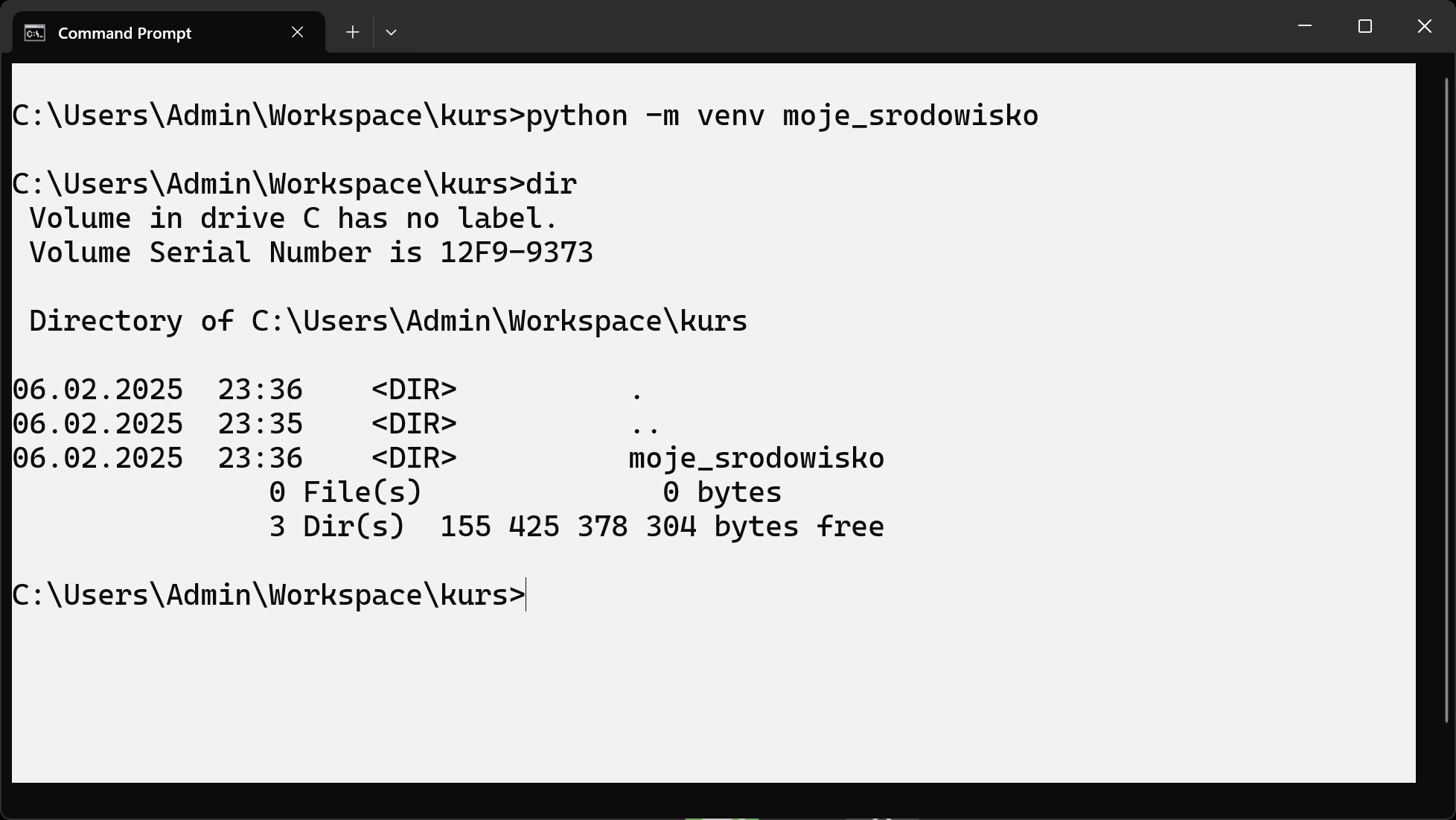
Rysunek . Przykładowy efekt zadziałania polecenia wyświetlenia wersji języka Python (win)

### Tworzenie wirtualnych środowisk

Aby stworzyć wirtualne środowisko języka Python, posłużymy się modułem *venv*. W tym celu należy rozpocząć od otwarcia terminala i przejścia do katalogu, w którym ma zostać utworzone środowisko, wykorzystując komendę zmiany katalogu, na przykład *cd /ścieżka/do/katalogu*. Po wejściu do odpowiedniego folderu, uruchamia się proces tworzenia wirtualnego środowiska, wpisując komendę *python -m venv moje\_srodowisko*

Listing 1.4 Komenda wywołująca stworzenie środowiska wirtualnego dla języka Python

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

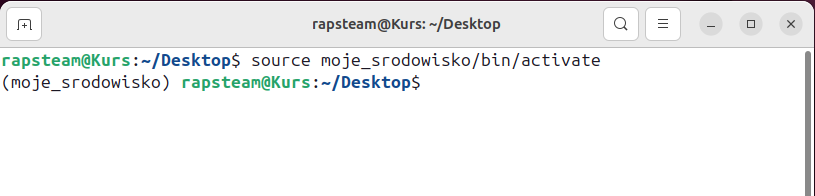


Rysunek . Efekt stworzenia środowiska wirtualnego dla języka Python o nazwie "moje\_srodowisko"

gdzie *moje\_srodowisko* stanowi wybraną nazwę dla nowego środowiska. W wyniku wykonania tej komendy powstaje katalog zawierający wszystkie niezbędne pliki, które umożliwiają izolację zależności i konfiguracji projektu od globalnej instalacji Pythona. Po utworzeniu środowiska należy je aktywować. W systemach opartych na Uniksie (takich jak Linux lub macOS) aktywację wykonuje się poleceniem *source nazwa\_env/bin/activate*, natomiast w systemie Windows należy użyć polecenia *moje\_srodowisko\Scripts\activate*.

Listing . Komenda aktywująca interpreter języka Python ze środowiska wirtualnego (linux)

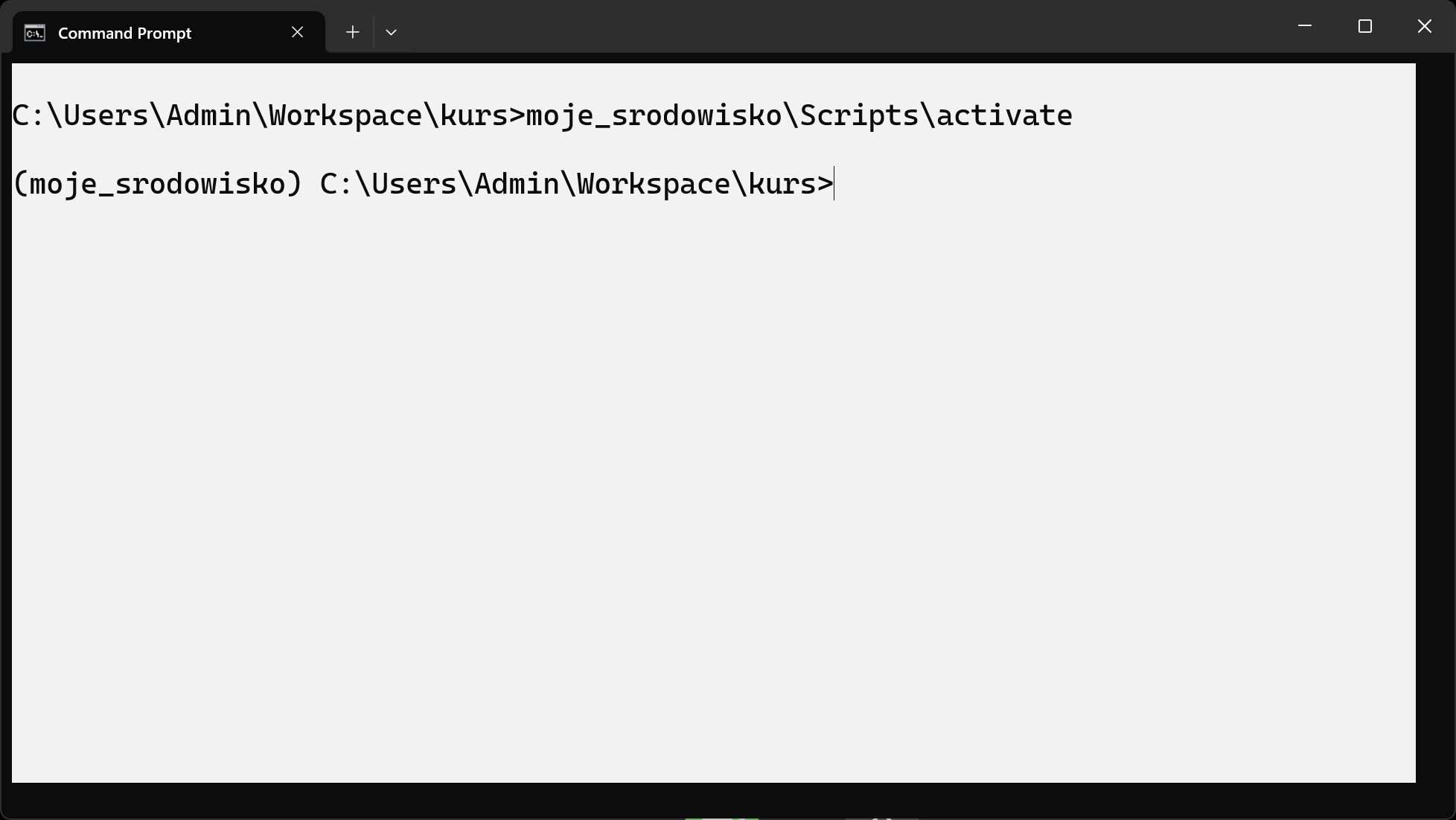
|  |
| --- |
| source moje\_srodowisko/bin/activate |



Rysunek . Efekt aktywacji środowiska wirtualnego (linux)

Listing . Komenda aktywująca interpreter języka Python ze środowiska wirtualnego (win)

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

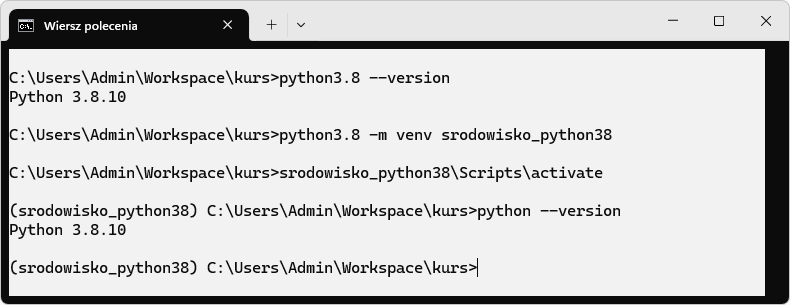


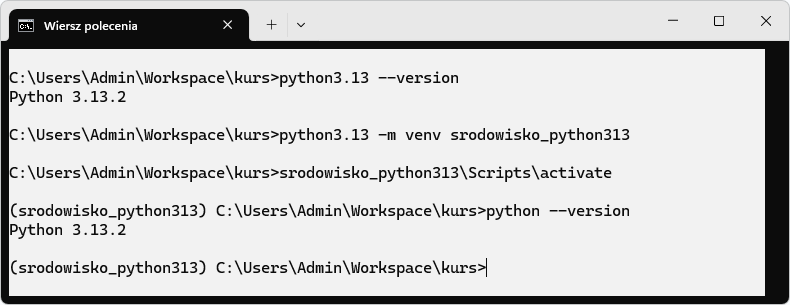
Rysunek . Efekt aktywacji środowiska wirtualnego (win)

Zmiana wyglądu promptu terminala potwierdza, że środowisko zostało aktywowane i jest gotowe do pracy. Poza aktywacją, można je w analogiczny sposób deaktywować poprzez użycia słowa *deactivate.*

### Zarządzanie wersjami

Zarządzanie wersjami Pythona jest istotne, gdyż projekty mogą wymagać korzystania z określonych wersji interpretera oraz specyficznych zestawów bibliotek, co wynika z ewolucji języka, zmian w bibliotece standardowej oraz zależności zewnętrznych pakietów. W tym celu poznanie sposobu tworzenia i działania środowisk wirtualnych za pomocą modułu *venv* okazuje się bardzo przydatne i usprawnia pracę. Utrzymanie oddzielnych środowisk dla poszczególnych projektów pozwala uniknąć konfliktów między wersjami oraz zapewnia, że każda aplikacja korzysta z odpowiedniej konfiguracji, bez wpływu na globalną instalację Pythona. Izolacja ta umożliwia również testowanie kodu w różnych wersjach interpretera, co przyczynia się do poprawy stabilności oraz przewidywalności działania aplikacji. Dlatego bazując na zainstalowanych w systemie interpreterach języka Python można tworzyć środowiska wirtualne bazując na wersjach na których zostały stworzone i które nie powodują kolizji ze sobą w systemie i tworzonych projektach.





Rysunek . Przykład utworzenia różnych środowisk wirtualnych z różnymi wersjami interpretera Python

### Zadania

**Zadanie 1.2.1 Tworzenie i aktywacja środowiska wirtualnego**

* W podanej przez prowadzącego ścieżce, utwórz wirtualne środowisko Python o nazwie *KursPython* i je aktywuj.

**Zadanie 1.2.2 Instalacja specyficznej wersji języka Python**

* Zainstaluj dowolną, inną wersję Python’a, niż ta przedstawiona na kursie

**Zadanie 1.2.3 \* Poznanie innego sposobu na zarządzanie wersjami języka**

* Jakie są inne metody/menadżery zarządzania wersjami interpretera Python?
* Jakie są ich plus i minusy względem prezentowanego rozwiązania?

**Zadanie 1.2.4\* Wykorzystanie innego modułu do zarządzania środowiskami**

* Stwórz środowisko wirtualne języka Python za pomocą modułu innego niż venv

**Zadanie 1.2.5\***

* Stwórz środowisko wirtualne za pomocą dowolnego menadżera języka Python.

## Zasady tworzenia i struktury skryptów w języku Python

Pliki z rozszerzeniem *.py* są podstawowymi jednostkami kodu w języku Python. Są to pliki tekstowe, które mogą zawierać instrukcje w języku Python, takie jak definicje funkcji, klasy, zmienne oraz importy modułów. Pliki *.py* można je edytować w dowolnym edytorze tekstowym. Znaczenie poszczególnych elementów kodu 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 .. Przykładowy plik języka Python .py z poprawnymi wcięciami

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

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

Listing .. Skrypt języka Python załamaniami lini

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

### Pierwsze skrypty w edytorze tekstowym

Skrypty języka Python są plikami tekstowymi, dlatego można je edytować w dowolnym edytorze tekstowym, 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 .. Program wsypujący „Witaj w świecie Pythona!”

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

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

### Uruchamianie skryptów Python w konsoli systemowej

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

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

Listing .. 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 .. Kod umożliwiający przekazywanie argumentów w wierszu poleceń

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

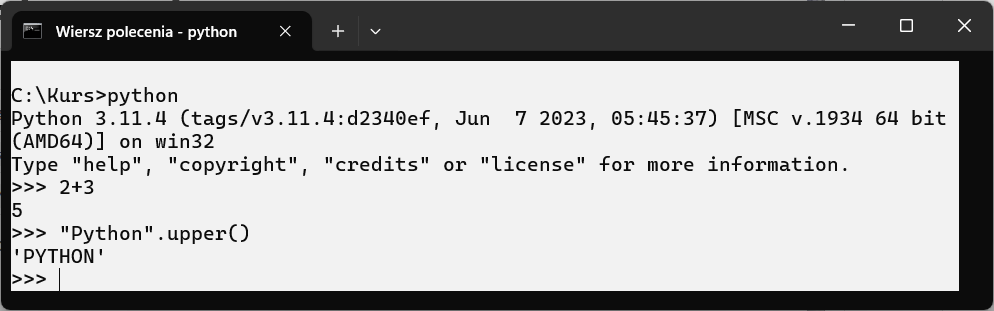
### Wybór trybu pracy: interaktywny czy skryptowy?

W **rozdziale 1** omówiliśmy wady i zalety korzystania z trybu interaktywnego oraz skryptowego. Teraz przeanalizujemy praktyczne zastosowania obu trybów oraz przedstawimy sytuacje, w których wybór właściwego trybu jest szczególnie istotny.

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

**Sprawdzania działania prostych komend**

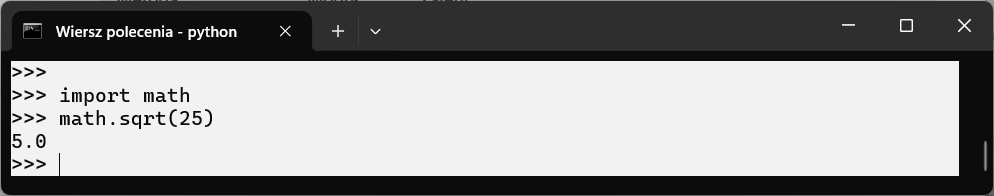
W przypadkach, gdy nie ma potrzeby tworzenia całego skryptu, warto najpierw sprawdzić poprawność pojedynczej instrukcji.



Rysunek .. 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 .. Szybka eksploracja modułów i funkcji

Tryb skryptowy pozwala na zapisanie kodu w pliku *.py*, co umożliwia jego wielokrotne uruchamianie bez konieczności każdorazowego wpisywania poleceń. Jest to kluczowe w przypadku pisania 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 .. Prosty kod, który można uruchamiać wielokrotnie

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

**Automatyzacja zadań – przykład sprawdzania daty codziennie**

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

Listing .. Kod automatyzujący zapis aktualnej daty i godziny

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

Tabela .. 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.

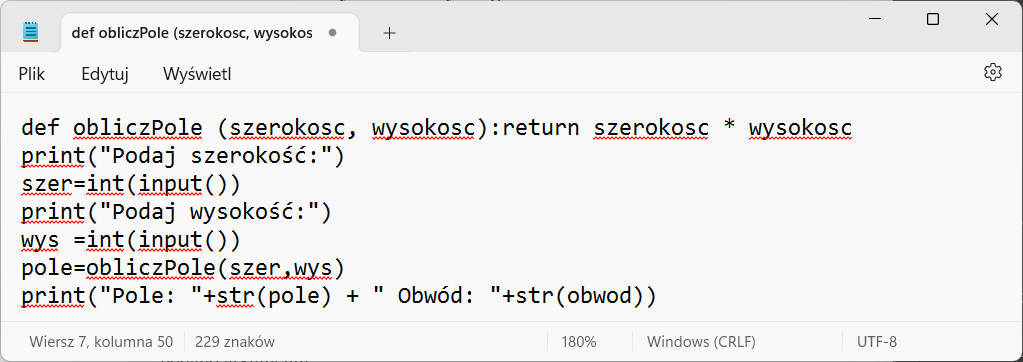
### Zadania podsumowujące temat (TYLKO SKRYPTY!)

**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 1.3.2. \* 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**

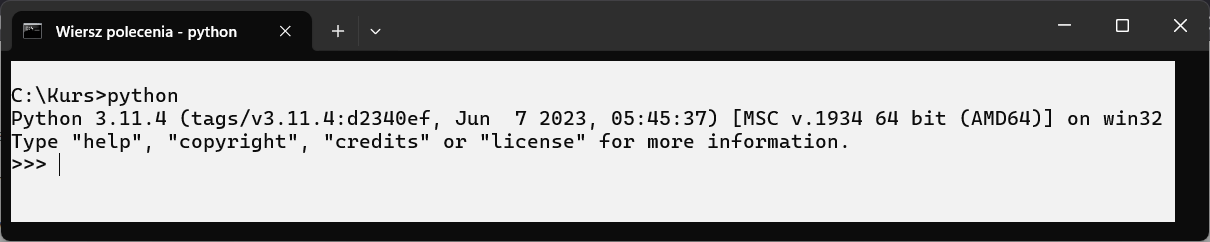
* Napisz skrypt, który pobiera aktualną datę i godzinę i zapisuje ją do pliku *log.txt*.
* Przykładowy format zapisu *2025-02-05 14:30:12 - Uruchomiono skrypt*
* Jeśli plik *log.txt* nie istnieje, skrypt powinien go utworzyć.

## Uruchamianie kodu w konsoli za pomocą interpretera

W tym rozdziale skupimy się na trybie interaktywnym Pythona, 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.



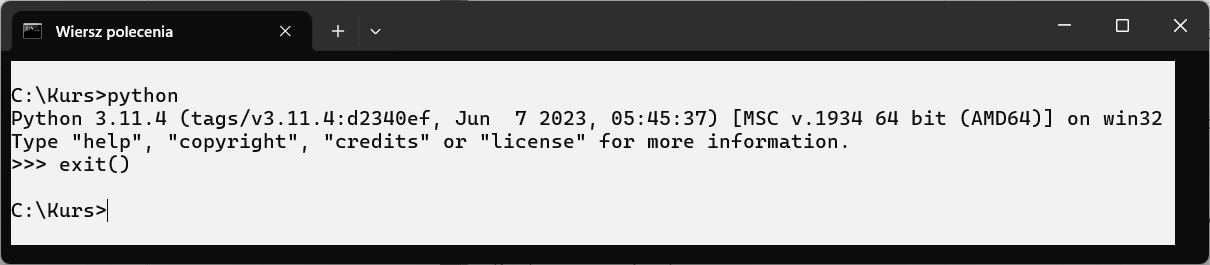
Rysunek .. Uruchomienie trybu interaktywnego dla domyślnej wersji Python

Jeśli chcemy uruchomić konkretną wersję Pythona, musimy najpierw 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 .. Uruchomienie Python 3.10

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

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

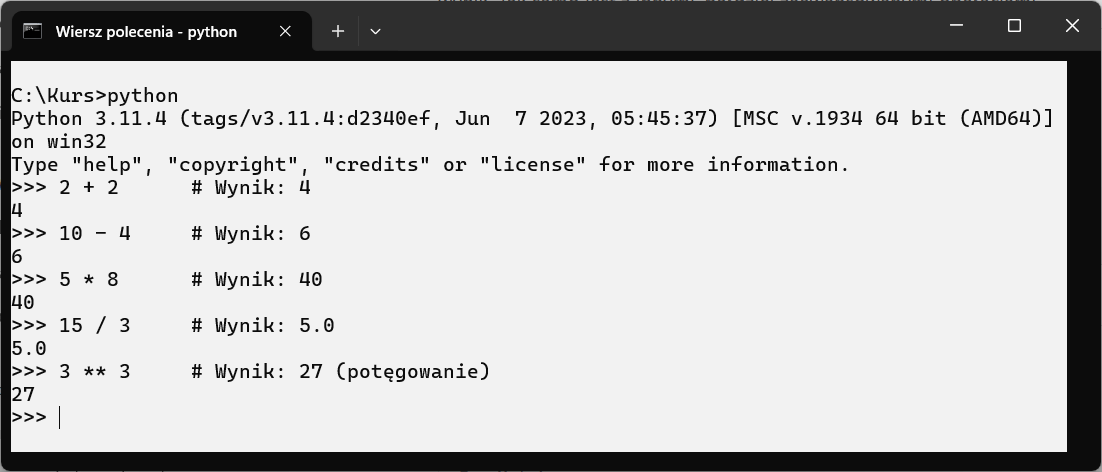


Rysunek .. Wyłączenie trybu interaktywnego

### Wprowadzanie pojedynczych komend i natychmiastowe obserwowanie wyników

Tryb interaktywny Pythona 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 .. Natychmiastowe obserwowanie wyników

### 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 .. Wielolinijkowy fragment kodu

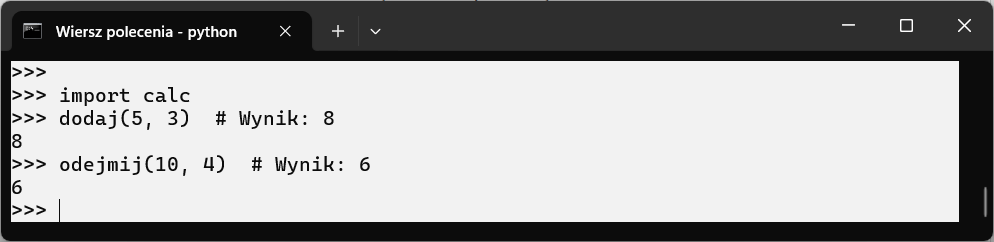
|  |
| --- |
| [module\_1\_lesson\_3\_calc.py] |

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



Rysunek .. Wielolinijkowy fragment kodu w trybie interaktywnym

Ten sam fragment kodu może zostać uruchomiony w trybie interaktywnym. To podejście zapewnia wielokrotne testowanie i ponowne użycie funkcji bez konieczności ich przepisywania.



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

### Zadania podsumowujące temat

**Zadanie 1.4.1. Uruchomienie wybranej wersji Python**

* Otwórz Wiersz polecenia (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**

* Zaimportuje bibliotekę: *import math*
* Sprawdź działanie *math.sqrt(25).*

**Zadanie 1.4.4. \***

* 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. \***

*Cel: Celem zadania jest zrozumienie, jak importować i ponownie ładować zmodyfikowane moduły w trybie interaktywnej konsoli Pythona 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 Pythonie 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 .. 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 wListing 1.10. Przykłady podstawowych operacji arytmetycznych, gdzie zaprezentowano ich praktyczne zastosowanie w języku Python.

Listing .. Przykłady podstawowych operacji arytmetycznych

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

**Priorytet operatorów matematycznych**

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

Listing .. Przykład użycia operatorów skróconych dodawania oraz potęgowania

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

**Operacje matematyczne na różnych typach danych**

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

Listing .. Operacje matematyczne na liczbach całkowitych i zmiennoprzecinkowych

|  |
| --- |
| [module\_1\_lesson\_3\_math\_operations\_types.py] |

### Manipulacja napisami

Napisy (łańcuchy znaków) w języku Python pozwalają na różnorodne operacje, takie jak łączenie operatorem +, powielanie za pomocą operatora \* oraz zmiana wielkości liter. Dzięki wbudowanym metodom i operatorom Python 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 Pythonie można to zrobić za pomocą operatora *+, join(), f-stringów* lub metody *format()*.

Tabela .. 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 .. Przykłady użycia metod łączenia napisów

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

**Powielanie napisów**

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

Tabela .. 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 .. Przykład użycia powielania napisów

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

**Zmiana wielkości liter**

Python oferuje wbudowane metody do zmiany wielkości liter w napisach. Tabela 1.1 przedstawia wbudowane metody Pythona umożliwiające zmianę wielkości liter w napisach, takie jak upper(), lower(), title() i capitalize(). Dzięki nim można łatwo konwertować tekst do wielkich lub małych liter oraz formatować nagłówki i zdania zgodnie z wymaganiami.

Tabela .. 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 .. Przykład użycia metod zmiany wielkości liter

|  |
| --- |
| [module\_1\_lesson\_3\_text\_case\_methods.py] |

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

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

Tabela .. Wykorzystanie funkcji wbudowanych len() i type()

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

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

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

### Zadania 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 (len()).

**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()).

## Poznanie typów zmiennych w Pythonie

Na początek powiedzmy czym jest zmienna. Zmienna to abstrakcyjny symbol, który reprezentuje określony obszar pamięci przeznaczony do przechowywania danych. Nazwa zmiennej służy jako etykieta umożliwiająca odwoływanie się do tej pamięci w trakcie wykonywania programu. Wartość przechowywana w zmiennej może ulegać zmianom, co umożliwia dynamiczne operacje na danych oraz elastyczne przetwarzanie informacji. W kontekście programowania zmienne są podstawowym narzędziem do przechowywania wyników obliczeń, danych wejściowych lub innych wartości niezbędnych do realizacji algorytmów. W praktyce przypisanie wartości do zmiennej odbywa się poprzez operator przypisania, co powoduje, że dany fragment pamięci zostaje oznaczony wybraną nazwą, a przypisana wartość staje się dostępna pod tą nazwą. W przypadku języka Python, takim symbolem przypisania jest znak *równa się* „=”. Po jego lewej stronie, określamy naszą etykietę, np. *a* natomiast po prawej, wartość jaką chcemy przypisać, np. *5.*

Listing 1.14 Przykład przypisania wartości 5 do zmiennych o nazwie (etykiecie) 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 Pythonie 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.

Python udostępnia wiele wbudowanych typów danych, które można podzielić na kilka głównych kategorii.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hierarchia typów standardowych w Pythonie | | | | |
| Liczbowe | Sekwencyjne | Zbiory | Mapujące | Braku wartości |
|  |  |  | - | - None |

1. Typy liczbowe
   1. Liczby całkowite
   2. Liczby zmiennoprzecinkowe
   3. Liczby zespolone
2. Wartości logiczne
   1. Tyb bool
3. Typy sekwencyjne
   1. Łańcuch znaków
   2. Listy
   3. Krotki
   4. Typ sekwencji bajtów
4. Typy zbiorów
   1. Zbiór
   2. Słownik
5. Typ braku wartości

### Typy liczbowe

Typy liczbowe w Pythonie stanowią podstawową kategorię danych służących do reprezentacji wartości liczbowych. W standardowej dystrybucji Pythona 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 dokonywać konwersji typów *float, str* i *bool* na typ całkowity *int*.

Listing 1.15 Przykłady liczb całkowitych

|  |
| --- |
| >>> a = 15 # to jest liczba całkowita  >>> b = -10243 # to też jest liczba całkowita  >>> c = a + b # to także jest liczba całkowita  >>> c  -10228  >>> type (c)  <class 'int'> |

**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. Za pomocą funkcji *float()* można dokonywać konwersji typów *int, str* i *bool* na typ zmiennoprzecinkowy *float.*

Listing 1.16 Przykłady liczb zmiennoprzecinkowych

|  |
| --- |
| >>> a = 1.0 # to jest liczba zmiennoprzecinkowa  >>> b = 3 # to nie jest liczba zmiennoprzecinkowa  >>> c = a / b # to jest liczba zmiennoprzecinkowa  >>> type(c)  <class 'float'>  >>> c  0.3333333333333333 # przykład zaokrąglenia liczby okresowej |

Listing 1.11.7 Przykład błędu zaokrąglenia liczb zmiennoprzecinkowych

|  |
| --- |
| >>> a = 0.1  >>> b = 0.2  >>> c = a + b # suma dwóch liczb typu float  >>> c  0.30000000000000004 # błąd zaokrąglenia, wynikający ze sposobu zapisu liczby float |

**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*.

Listing 1.11.8 Przykład użycia liczby zespolonej

|  |
| --- |
| >>> z = 3 + 4j # to jest liczba zespolona  >>> type(z)  <class 'complex'>  >>> z.real # wyświetla część rzeczywistą  3.0  >>> z.imag # wyświetla część urojoną  4.0  >>> abs(z) # wyświetla moduł liczby zespolonej  5.0 |

### Wartość logiczna

Wartość logiczna, inaczej *bool*, stanowi typ wbudowany 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 ściśle powiązany z typem *int*, przy czym wartość *true* odpowiada liczbie 1 a *false* liczbie 0, co umożliwia pewne operacje arytmetyczne. Nie jest to jednak zalecane w kontekście używania wartości logicznych. Mogą być one generowane na skutek działania operacji porównawczych, logicznych czy funkcji warunkowych jak również konwersji z innych typów danych za pomocą funkcji wbudowanej *bool()*, która interpretuje obiekty puste lub zero jako *false* a pozostałe jako *true*. Za pomocą tej funkcji można dokonywać konwersji typów *int, float, str, bool* i zbiory typu *list, dict* i *tuple.*

Listing .1. Przykład deklaracji typu logicznego i jego generacja na wskutek operacji porównania

|  |
| --- |
| >>> condition = True  >>> type(condition)  <class 'bool'>  >>> a = 3  >>> b = 4  >>> c = (a == b) # przypisanie wartości logicznej po operacji porównania liczb  >>> type(c)  <class 'bool'>  >>> c  False |

### Typy sekwencyjne

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

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

**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.*

Listing 1.20 Przykład stworzenia ciągu znaków w postaci zdania, oraz pojedynczego znaku

|  |
| --- |
| >>> tekst = "Ala ma kota" # przypisanie zdania jako ciągu znaków  >>> type(tekst)  <class 'str'>  >>> tekst  'Ala ma kota'  >>> znak = 'C' # przypisanie znaku, czyli ciągu o długości 1  >>> type(znak)  <class 'str'> |

**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 (czyt. pierwszy element w liście ma indeks 0)

Listing 1.21 Przykład utworzenia listy zawierającej wiele typów danych,

|  |
| --- |
| >>> lista = [5, 0.3, 'c', False] # utworzenie listy składającej się z wielu typów  >>> type(lista)  <class 'list'>  >>> tekst = "Ala ma kota"  >>> napis = list(tekst) # zamiana łańcucha znaków na listę  >>> napis  ['A', 'l', 'a', ' ', 'm', 'a', ' ', 'k', 'o', 't', 'a'] # lista znaków  >>> napis[7] # wyświetlenie 8’ego elementu  'k' |

**Krotki (tuple)**

Są niemutowalnymi sekwencjami podobnymi do list. Po ich utworzeniu zawartość krotki nie może zostać zmieniona, co sprawia, że są one przydatne w sytuacjach wymagających zachowania stałego zestawu danych. Krotki zapisuje się za pomocą nawiasów okrągłych lub poprzez rozdzielenie elementów przecinkami bez nawiasów. Niemutowalność krotek umożliwia ich wykorzystanie jako kluczy w słownikach lub elementów zbiorów, pod warunkiem że wszystkie ich składniki również są niemutowalne. Dodatkowo w krotki można opakować więcej niż jeden wynik działania funkcji.

Listing 1.22 Przykład utworzenia krotki oraz stowrzenia jej jako zwracany typ funkcji

|  |
| --- |
| >>> coords = (52.439898, 21.013342)  >>> type(coords)  <class 'tuple'>  >>> coords[0]  52.439898  >>> def calculate(a, b):  … suma = a + b  … iloczyn = a \* b  … return (suma, iloczyn)  >>> wynik = calculate(3, 4)  >>> wynik  (7, 12)  >>> coord[0] = 50.233 # Błąd, elementu tupli nie można zmieniać |

**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.

Listing 1.22 Przykład utworzenia krotki oraz stowrzenia jej jako zwracany typ funkcji

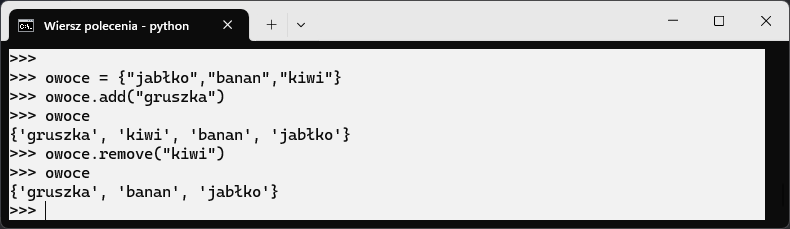
|  |
| --- |
| >>> data = bytearray('hello', 'utf-8')  >>> data  bytearray(b'hello')  >>> data[0] = 72 # 72 w kodzie ascii do duża litera 'H'  >>> data  bytearray(b'Hello') |

### Zbiory

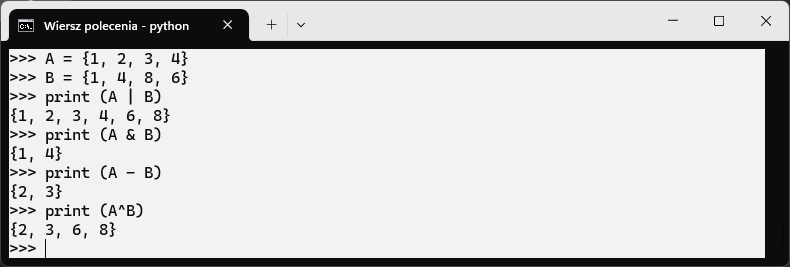
Zbiory w Pythonie umożliwiają przechowywanie nieuporządkowanych zbiorów unikalnych elementów.

**Zbiór (set)**

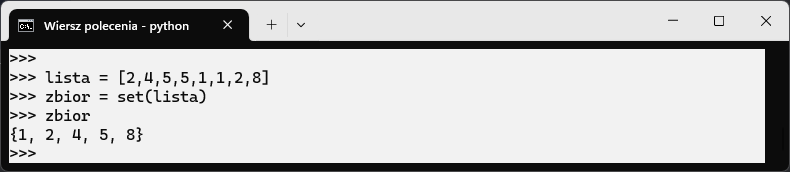
Tworzony jest poprzez ujęcie elementów w nawiasy klamrowe, na przykład: *{1, 2, 3},* lub za pomocą funkcji *set(),* która konwertuje iterowalne obiekty na zbiór. *Set* umożliwia wykonywanie operacji matematycznych, takich jak suma, różnica, przecięcie czy symetryczna różnica, co przydaje się przy analizie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



Rysunek . Operacja utworzenia zbioru i dodania i usunięcia jego elementu



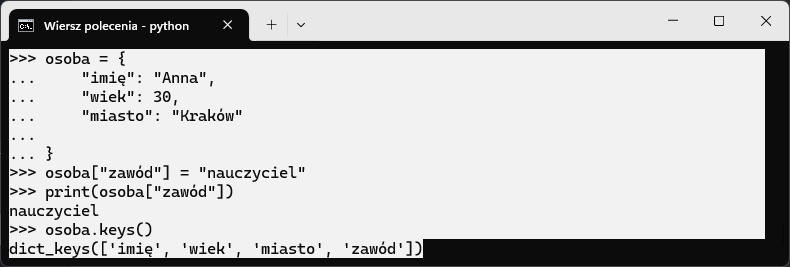
Rysunek . Przykłady operacji na zbiorach



Rysunek . Przykład usunięcia duplikatów i posortowania elementów listy poprzez rzutowanie zmiennej na zbiór

**Słownik (dictionary)**

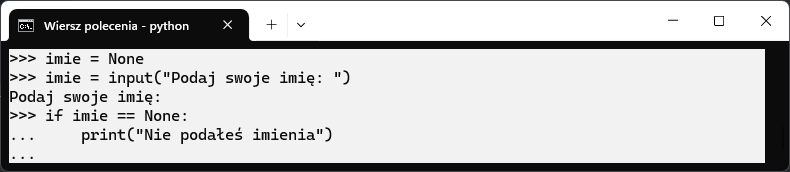
Reprezentują strukturę danych opartą na parach klucz-wartość, w której każdy klucz jest unikalny i musi być typem niemutowalnym. Słownik umożliwia szybki dostęp do danych na podstawie przypisanego do nich klucza Elementy słownika tworzone są w postaci par, gdzie klucz i odpowiadająca mu wartość oddzielone są dwukropkiem, a całość zamknięta jest w nawiasach klamrowych. Możliwość dynamicznego dodawania, modyfikowania i usuwania par klucz-wartość sprawia, że słowniki są wszechstronnym narzędziem do przechowywania i wyszukiwania danych. Struktura ta jest wykorzystywana w wielu algorytmach, w których istotny jest szybki dostęp do informacji i efektywne przetwarzanie danych.



Rysunek . Przykład utworzenia słownika, dodania pola "zawód" i wyświetlenie kluczy jakie zawiera

### Typ braku wartości

NoneType to wbudowany typ danych w Pythonie reprezentujący brak wartości. Pełni funkcję symbolu określającego nieistnienie lub brak przypisanej wartości. W Pythonie None jest używane między innymi jako domyślny wynik funkcji, które nie zwracają jawnie innej wartości, oraz jako wskaźnik na niezainicjowaną zmienną lub wynik operacji, której wynik nie został określony. NoneType i jego jedyny obiekt None stanowią standardowy sposób reprezentacji braku danych lub stanu nieustalenia w aplikacjach, zabezpieczając w ten sposób nasz system przed nieprawidłowym dostępem do pamięci przez aplikację. Nadpisanie rejestrów spoza zarezerwowanej przez aplikację pamięci, w najlepszym przypadku skończyłoby się zaprzestaniem działania programu, a w najgorszym, uszkodzeniu naszego systemu.



Rysunek . Przykład utworzenia zmiennej typu None, sprawdzenie jej i wyświetlenia komunikatu i błędzie jeżeli pozostała pusta

### 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:
  + Zmienne typów *str* na *int, float, bool* i *complex* oraz vice versa

**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?

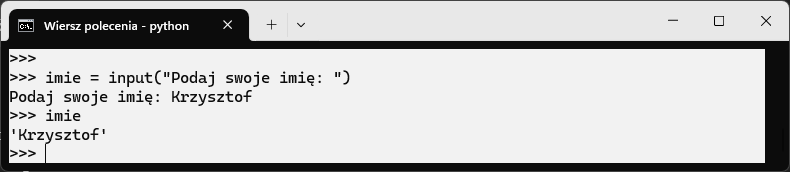
**Zadanie 1.6.5. \***

* Czy jest jakiś sposób, aby zmienić wartość zmiennych określanych jako niemutowalnych?

## Używanie metod wejścia i wyjścia

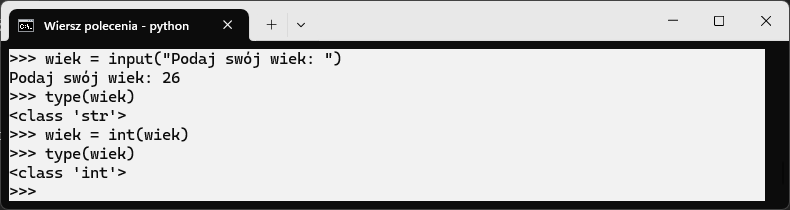
### Funkcja input()

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 . Pobranie od użytkownika danych za pomocą funkcji input()

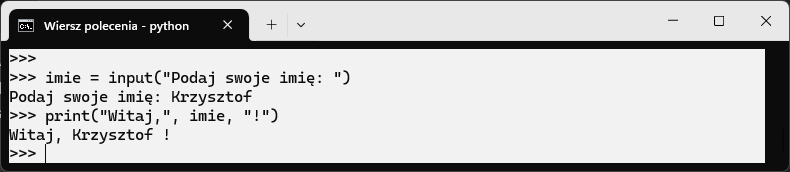
Warto zauważyć, że funkcja *input()* zawsze zwraca wartość typu string, co oznacza, że przy pobieraniu liczb konieczna jest konwersja, na przykład przy użyciu funkcji *int()* lub *float()*:



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

### Funkcja print()

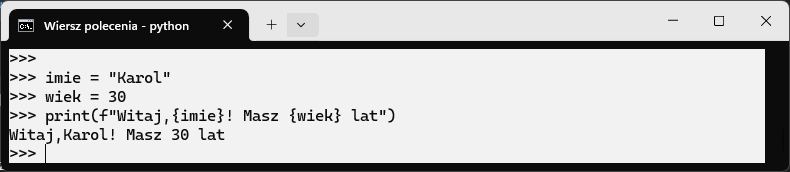
Funkcja *print()* służy do prezentowania wyników na ekranie poprzez standardowe wyjście. Argumenty przekazywane do funkcji są konwertowane na ciągi znaków, a następnie wyświetlane. Domyślnie kolejne argumenty oddzielane są spacją, a na końcu wyniku funkcja dodaje znak nowej linii. Przykładowe użycie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

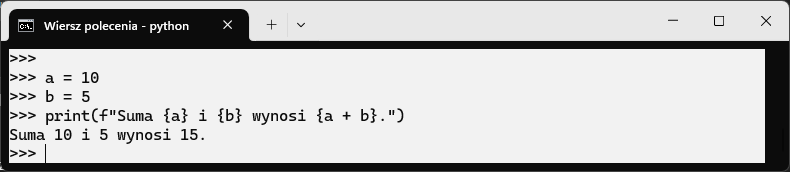
### Formatowanie stringów

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 . 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 . 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]."*

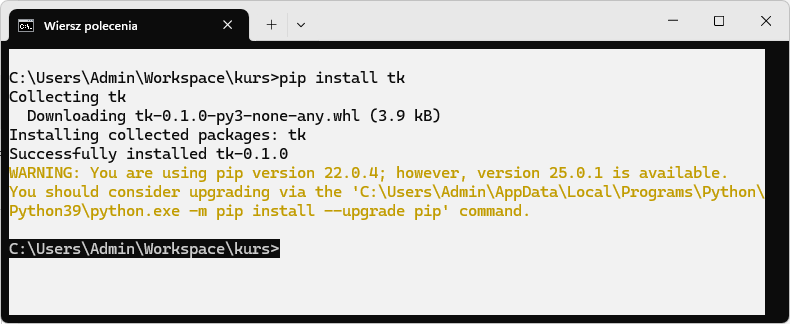
## Korzystanie z bibliotek i menadżera PIP

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

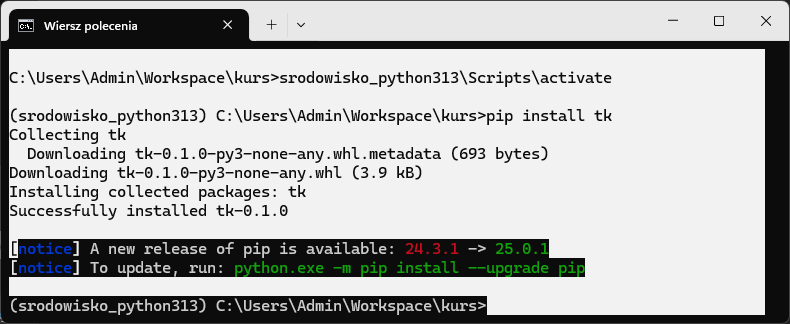
### Instalacja i wykorzystanie bibliotek z pomocą narzędzia PIP

PIP to menadżer pakietów, umożliwiający instalację oraz zarządzanie zewnętrznymi bibliotekami, które zwiększają funkcjonalność Pythona. Aby znaleźć bibliotekę, można odwiedzić stronę PyPI (Python Package Index) pod adresem https://pypi.org/, gdzie dostępne są tysiące pakietów. Instalacja biblioteki odbywa się za pomocą polecenia *pip* wpisywanego w terminalu naszego systemu.



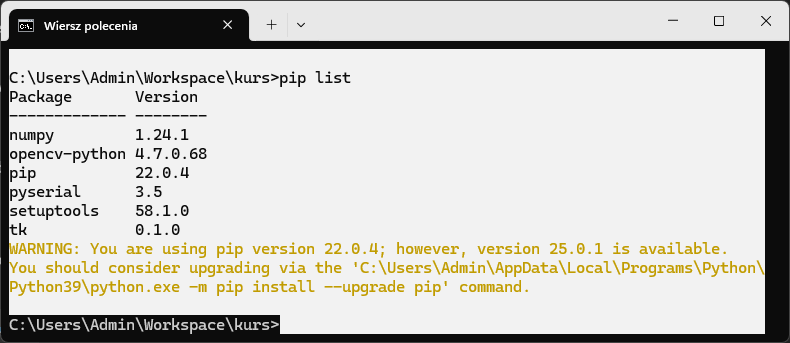
Rysunek . Przykład instalacji biblioteki tkinter na poziomie globalnym

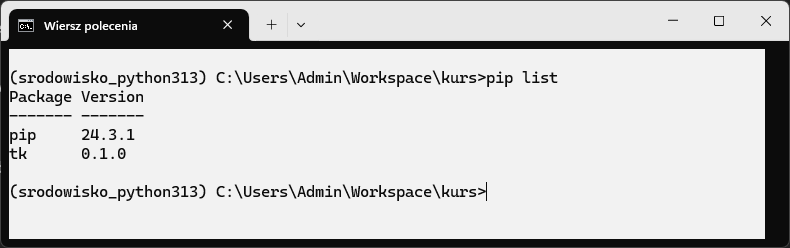
Instalowanie bibliotek za jego pomocą jest bardzo wygodne i szybkie. Należy jednak mieć na uwadze fakt, że wykonywanie go na poziomie systemu, powoduje, że biblioteka jest instalowana globalnie i może to powodować konflikty z innymi modułami korzystającymi z innych zależności. Dlatego też, przy dużych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

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





Rysunek . Przykład wyświetlenia zawartości zainstalowanych bibliotech 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.

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

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

|  |
| --- |
| pip list |

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

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

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

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

### Zadania

//

- instalacja sprawdzenie wersji turle

- narysować koło *zolw.circle(50)*

* *Znajdz listę wersji tutrle w PIP i ziasntaluj z ltuerog*

//

**Zadanie 1.8.1.**

**Zadanie 1.8.2.**

**Zadanie 1.8.3.**

**Zadanie 1.8.4. \***

**Zadanie 1.8.5. \***

## Podsumowanie modułu

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.

Ten rozdział pozwolił zdobyć wiedzę..

## Zadania podsumowujące moduł

**Zadanie 1.10.1.**

Zainstaluj Python w wersji 3.11, bez usuwania poprzedniej wersji.

**Zadanie 1.10.2.**

Wykonaj to i to.

**Zadanie 1.10.3.**

Wykonaj to i to.

**Zadanie 1.10.4.**

Wykonaj to i to.

**Zadanie 1.10.5.**

Wykonaj to i to.

**Zadanie 1.10.6.**

Wykonaj to i to.

**Zadanie 1.10.7.**

Wykonaj to i to.

**Zadanie 1.10.8.**

Wykonaj to i to.

**Zadanie 1.10.9.**

Wykonaj to i to.

**Zadanie 1.10.10.**

Wykonaj to i to.

**Zadanie 1.10.11.**

Wykonaj to i to.

**Zadanie 1.10.12.**

Wykonaj to i to.

# 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.