UNIWERSYTET EKONOMICZNY W POZNANIU

UNIWERSYTET EKONOMICZNY W POZNANIU · KATEDRA INFORMATYKI EKONOMICZNEJ

Algorytmy i struktury danych

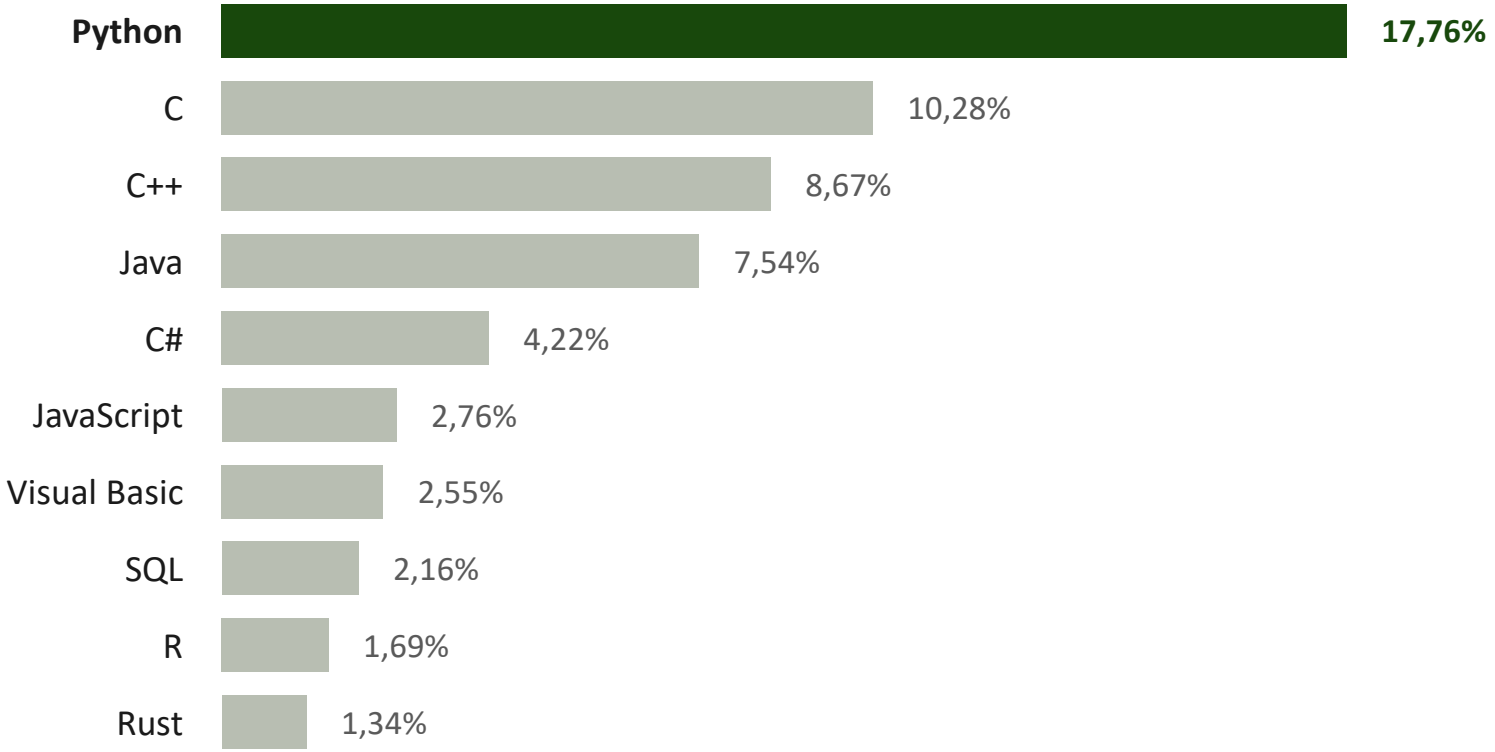
Zajęcia 1 · część 2: pierwsze kroki w Pythonie
rok akademicki 2026/2027 · dr inż. Szczepan Górtowski

Po tych zajęciach

- uruchamiasz komórki notatnika Jupyter i czytasz ich wynik
- tworzysz zmienne i rozpoznajesz ich typ
- liczysz operatorami i sterujesz programem przez if, elif, else
- piszesz funkcję i wiesz, kiedy print, a kiedy return
- piszesz kod, który przechodzi test z notatnika

Python prowadzi w indeksie TIOBE: 17,8%, drugi C — 10,3%

TIOBE liczy, jak często język pojawia się w wyszukiwarkach. To miara popularności, nie jakości.



CO Z TEGO MASZ

- biblioteki do danych, wykresów i AI: pandas, matplotlib, scikit-learn
- duża społeczność: łatwo znajdziesz przykłady i pomoc

UCZCIWIE

Python stracił 8,22 p.p. w rok, ale nad drugim C wciąż ma 7,48 p.p. przewagi.

TIOBE Index, wrzesień 2026, tiobe.com/tiobe-index (sprawdzone 7.10.2026).

Dziś: od komórki notatnika do funkcji, która przechodzi test

Kolejność jak w notatniku: sekcje 1–11.

01

Notatnik

komórki, kolejność, wynik

02

Zmienne

typy, listy, słowniki, stałe

03

Operatory

/, //, %, porównania, if

04

Funkcje

def, return, print

05

Styl

PEP 8, snake_case

06

Zadania

Hello World!, funkcje z testem

Notatnik to komórki: tekst, kod i wynik pod spodem

Pracujemy w pliku 01_Wprowadzenie_do_Pythona.ipynb. Uruchamiasz komórki po kolei, od góry.



1 **Komórka tekstu**

Markdown: nagłówki, opisy, zadania

2 **Komórka kodu**

Ctrl + Enter uruchamia, Shift + Enter uruchamia i idzie dalej

3 **Wynik**

pojawia się pod komórką, która go wypisała

Zrzut odtworzony z sekcji 6 notatnika; wynik policzony przez wykonanie komórki.

Ćw. 0 · Przewidź wynik, zanim uruchomisz

5 min

1. Nie uruchamiaj jeszcze komórki z operatorami.
2. Przy każdej linii zapisz, co wypisze Python.
3. Uruchom komórkę (Ctrl + Enter) i porównaj ze swoimi odpowiedziami.

KOMÓRKA Z SEKCJI 4 NOTATNIKA

```
print("1 < 2 ->", 1 < 2)
print("2 < 1 ->", 2 < 1)
print("1 + 1 =", 1 + 1)
print("3 - 1 =", 3 - 1)
print("2 * 7 =", 2 * 7)
print("121 / 11 =", 121 / 11)
print("121 // 12 =", 121 // 12)
print("10 % 3 =", 10 % 3)
```

PYTANIE

Ile wynosi 121 / 11? Liczba czy coś więcej? A 10 % 3?

Klucz do ćw. 0: dzielenie / zawsze daje float

Wyniki z uruchomienia komórki. Kolumna Typ to type(wynik) — sprawdzisz ją sam w notatniku.

Wyrażenie	Wynik	Typ
`1 < 2`	`True`	bool
`2 < 1`	`False`	bool
`1 + 1`	`2`	int
`3 - 1`	`2`	int
`2 * 7`	`14`	int
`121 / 11`	`11.0`	float
`121 // 12`	`10`	int
`10 % 3`	`1`	int

TRZY RODZAJE DZIELENIA

- / — dzielenie, wynik zawsze float: 121 / 11 = 11.0
- // — dzielenie całkowite, bez reszty: 121 // 12 = 10
- % — reszta z dzielenia: 10 % 3 = 1

ZAPAMIĘTAJ

Porównanie (<, ==, !=) zwraca bool: True albo False. To paliwo dla if.

01

Komentarze: piszesz je dla człowieka

to komentarz, a """ … """ to tekst — Jupyter go pokaże

Komórka z sekcji 1 notatnika. Blok w potrójnym cudzysłowie to łańcuch znaków, nie komentarz.

```
# Komentarz jednolinijkowy
PI = 3.14  # liczba pi — komentarz na końcu linii

"""
Blok komentarza (technicznie: łańcuch znaków), wygodny dla długich objaśnień,
list TODO itp.
"""
```

OUT — CO POKAŻE JUPYTER

```
'\nBlok komentarza (technicznie: łańcuch znaków), wygodny
dla długich objaśnień,\nlist TODO itp.\n'
```

KIEDY CO

- # — uwaga do jednej linii
- """ … """ — opis funkcji (docstring)
- linie do 79 znaków: bez „węży”

PUŁAPKA

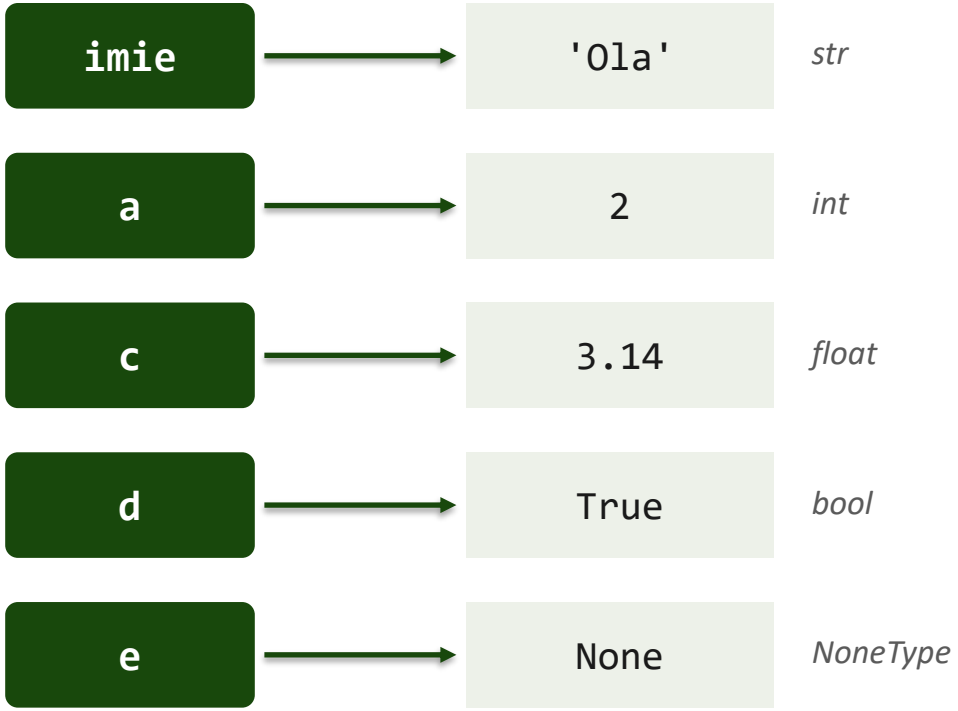
Blok """ na końcu komórki Jupyter wypisze. # nie wypisze nigdy.

02

Zmienne: etykiety na wartościach

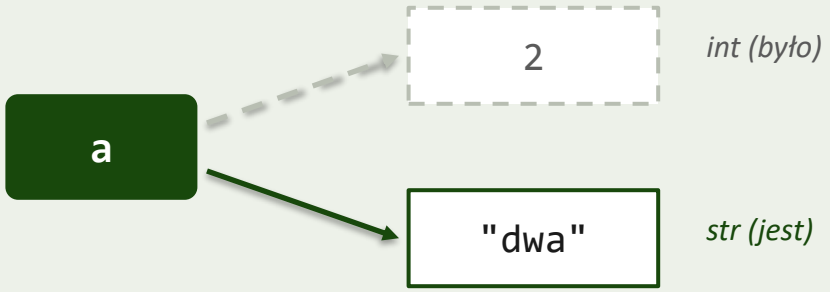
Zmienna to etykieta na wartości — typ ma wartość, nie nazwa

Python jest dynamicznie typowany: nie deklarujesz typu, a tę samą nazwę możesz przypiąć do innej wartości.



TA SAMA NAZWA, NOWA WARTOŚĆ

```
a = 2
a = "dwa"
```



8 zmiennych, 7 typów: Python rozpoznaje typ sam

Komórka z sekcji 2 i jej wynik. Liczby zmiennoprzecinkowe piszesz z kropką: 3.14, nie 3,14.

```
imie = "Ola"
a = 2
b = "Ola"
c = 3.14  # liczby zmiennoprzecinkowe zapisujemy z kropką
d = True
e = None

lista = [2, 4, 6, 0, [2, 4], "Ola"]
slownik = {"imie": "Ola", "funkcja": "nauczyciel"}

for nazwa, wartosc in {
    'imie': imie, 'a': a, 'b': b, 'c': c, 'd': d, 'e': e,
    'lista': lista, 'slownik': slownik
}.items():
    print(f"{nazwa:<8} -> {wartosc!r:<20} | typ: {type(wartosc).__name__}")
```

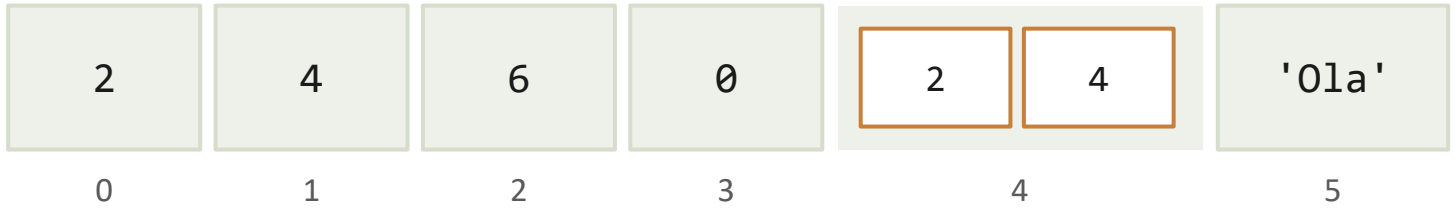
WYNIK KOMÓRKI

Nazwa	Wartość	Typ
imie	'Ola'	str
a	2	int
b	'Ola'	str
c	3.14	float
d	True	bool
e	None	NoneType
lista	[2, 4, 6, 0, [2, 4], 'Ola']	list
slownik	{'imie': 'Ola', 'funkcja': 'nauczyciel'}	dict

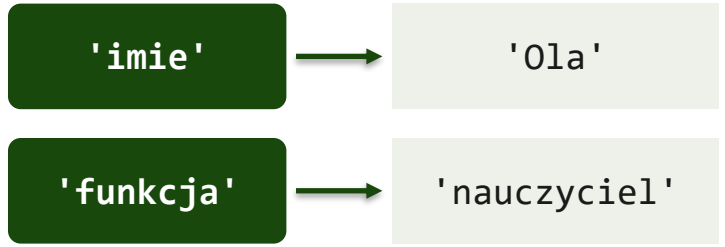
Lista pamięta kolejność, słownik — pary klucz: wartość

Obie struktury pochodzą z komórki z sekcji 2. Indeks listy zaczyna się od 0.

LISTA



SLOWNIK



SPRAWDŹ W NOTATNIKU

```
lista[0]    → 2
lista[4]    → [2, 4]
lista[4][1] → 4
lista[-1]   → 'Ola'
len(lista)  → 6
```

Lista może trzymać różne typy, także inną listę: lista[4] to cała lista [2, 4].

SPRAWDŹ W NOTATNIKU

```
slownik["imie"] → 'Ola'
slownik["funkcja"] → 'nauczyciel'
```

WIELKIE_LITERY to umowa: Python pozwoli nadpisać stałą

Sekcja 3 notatnika. Stała w Pythonie to konwencja dla ludzi, a nie blokada w języku.

```
MAX_ILOSC_ZAMOWIEN = 120  # konwencja: traktuj jak stałą
print("MAX_ILOSC_ZAMOWIEN =", MAX_ILOSC_ZAMOWIEN)
```

WYNIK

```
MAX_ILOSC_ZAMOWIEN = 120
```

PRZYKŁAD SPOZA NOTATNIKA

```
MAX_ILOSC_ZAMOWIEN = 150   # Python nie protestuje
print("MAX_ILOSC_ZAMOWIEN =", MAX_ILOSC_ZAMOWIEN)
```

WYNIK

```
MAX_ILOSC_ZAMOWIEN = 150
```

ZASADA

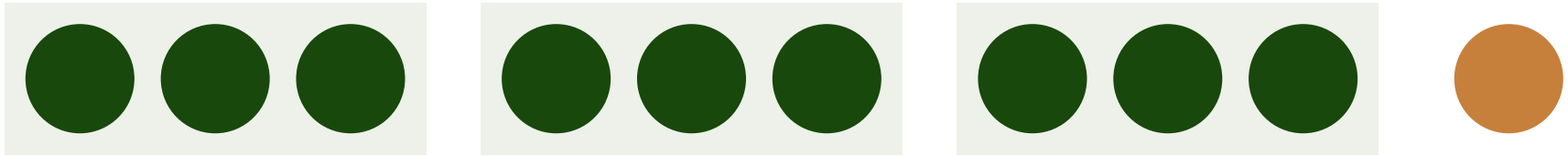
Nazwę WIELKIMI LITERAMI dajesz wartości, której nikt nie powinien zmieniać: limitom, stawkom, ścieżkom do plików.

03

Operatory i warunki: liczysz, porównujesz, wybierasz

10 // 3 = 3 i 10 % 3 = 1: trzy pełne trójki i reszta

Dzielenie całkowite mówi, ile pełnych grup się zmieści. Modulo — ile zostało.



grupa 1 | grupa 2 | grupa 3 | **reszta**

Wyrażenie	Wynik	Bo
`10 // 3`	3	10 = 3 × 3 + 1
`10 % 3`	1	zostaje 1
`121 // 12`	10	121 = 12 × 10 + 1
`7 % 2`	1	7 jest nieparzyste

DO CZEGO SIĘ PRZYDA

n % 2 == 0 sprawdza parzystość. To samo pytanie zadasz w algorytmach: czy liczba dzieli się przez k?

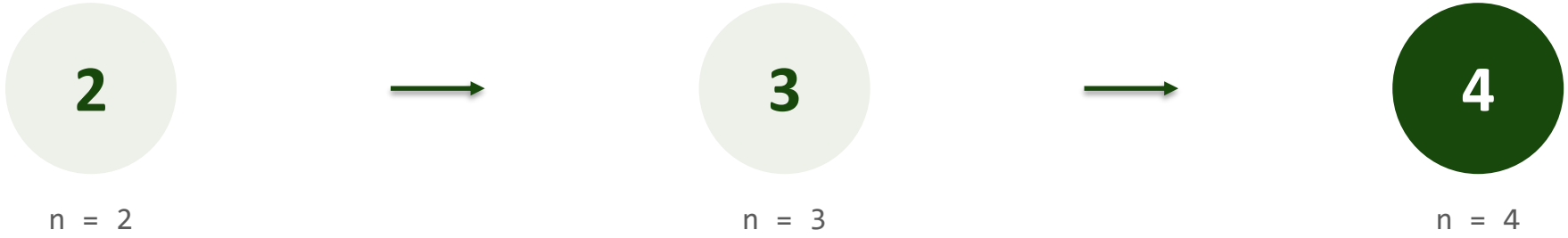
n += 1 to skrót n = n + 1: po dwóch krokach n = 4

Pierwsza część komórki z sekcji 5. Każda linia nadpisuje etykietę n nową wartością.

```
n = 2
```

```
n = n + 1
```

```
n += 1
```



POTĘGOWANIE

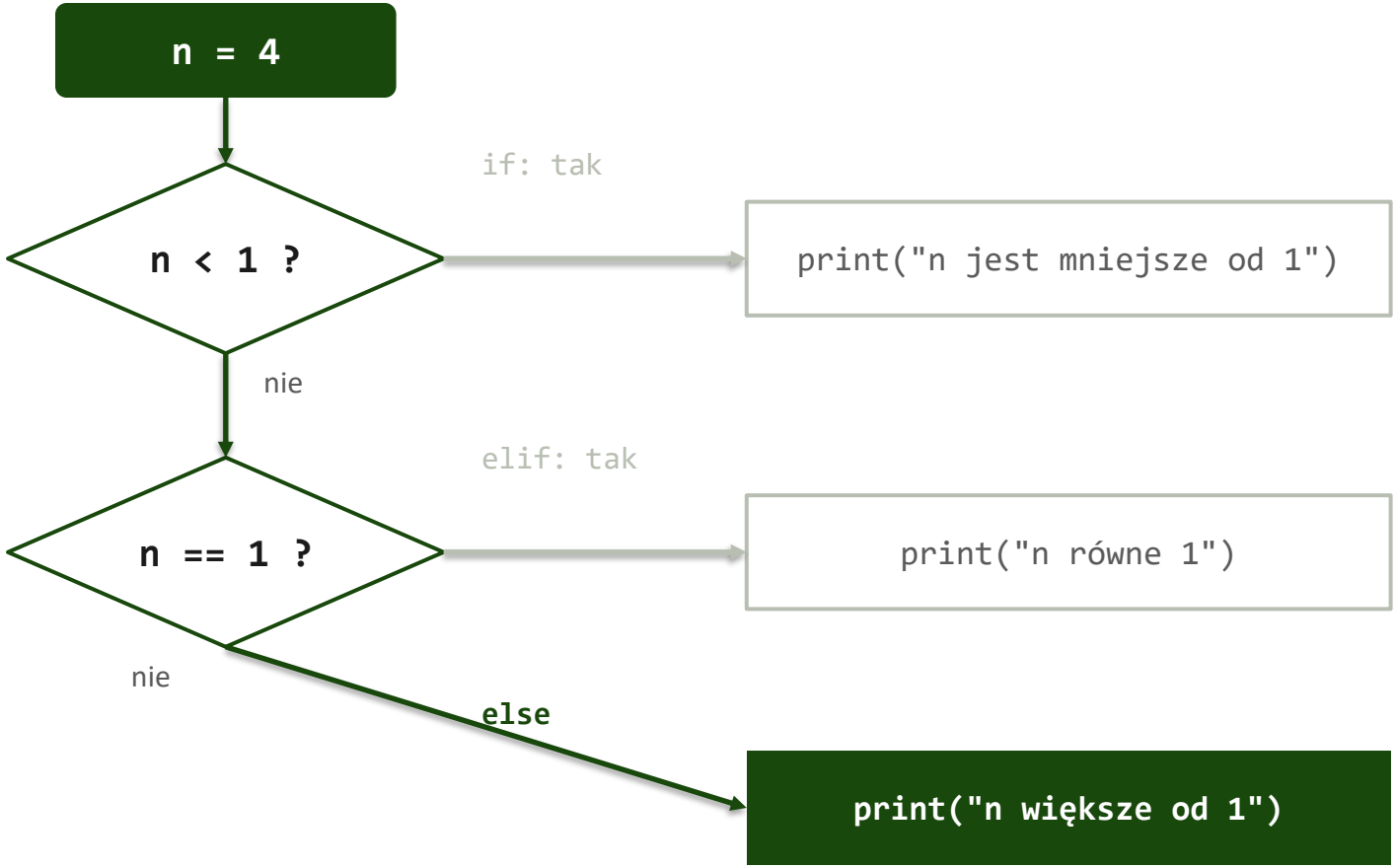
2 ** 4 = 16

TEN SAM WZÓR

-=, *=, /= działają tak samo: x *= 2 to x = x * 2

if, elif, else: dla n = 4 Python wybiera gałąź else

Druga część komórki z sekcji 5 jako schemat blokowy. Python sprawdza warunki od góry i wchodzi do pierwszej prawdziwej gałęzi.



WYNIK KOMÓRKI

```
n większe od 1
```

SKŁADNIA

- dwukropek kończy linię z warunkiem
- wcięcie (4 spacje) wyznacza, co należy do gałęzi
- elif może być wiele, else najwyżej jedno

04

Funkcje: nazwij raz, wołaj wiele razy

add_one(5) zwraca 6: funkcja w pięciu elementach

Komórka z sekcji 6. Definiujesz raz, wywołujesz ile razy chcesz.

```
1  def add_one(n):  2
3      """Zwraca n + 1."""
       n = n + 1
4      return n

5  print("add_one(5) ->", add_one(5))
```

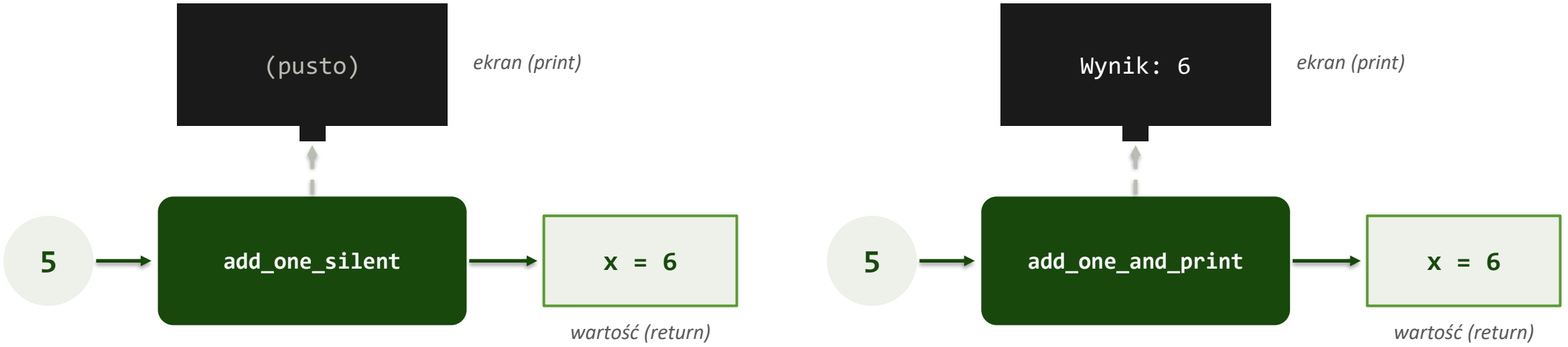
WYNIK

```
add_one(5) -> 6
```

1. **`def`**
   zaczyna definicję funkcji
2. **`add_one(n)`**
   nazwa w snake_case i parametr n w nawiasie
3. **`"""…"""`**
   docstring: co funkcja robi
4. **`return n`**
   oddaje wynik temu, kto wywołał
5. **`add_one(5)`**
   wywołanie: n dostaje wartość 5

print pokazuje człowiekowi, return oddaje wartość programowi

Dwie funkcje z sekcji 7 notatnika. Obie zwracają 6, ale tylko jedna coś wypisuje.



REGUŁA

Funkcja, której wynik ma iść dalej, kończy się return. print zostaw do podglądu i komunikatów.

Ostatnie 6 wypisał Jupyter, nie print

Komórka z sekcji 7 i jej pełny wynik. Jupyter pokazuje wartość ostatniego wyrażenia w komórce.

```python
def add_one_silent(n):
    return n + 1  # nic nie drukuje

def add_one_and_print(n):
    wynik = n + 1
    print("Wynik:", wynik)
    return wynik

# Wywołania:
x = add_one_silent(5)      # tu nic się nie wyświetli
print("Zmienna x zawiera:", x)
add_one_and_print(5)       # tu zobaczymy linijkę "Wynik: 6"
```

STDOUT — Z PRINT

```
Zmienna x zawiera: 6
Wynik: 6
```

OUT — OD JUPYTERA

```
Out: 6
```

UWAGA

W skrypcie .py ta trzecia linia się nie pojawi. Nie myl „Out” z print.

Funkcja bez return oddaje None — i y + 1 kończy się błędem

Przykład spoza notatnika: ta sama funkcja, ale z samym print.

```python
def add_one_only_print(n):
    print("Wynik:", n + 1)

y = add_one_only_print(5)
print("y =", y)
y + 1
```

WYNIK

```
Wynik: 6
y = None
```

BŁĄD

```
TypeError: unsupported operand type(s) for +:
  'NoneType' and 'int'
```

JAK TO NAPRAWIĆ

Zamień print na return n + 1. Teraz y = 6, a y + 1 = 7.

05

Styl: kod czytasz częściej, niż piszesz

„Kod czyta się znacznie częściej, niż się go pisze”

— PEP 8 — Style Guide for Python Code, G. van Rossum, B. Warsaw, A. Coghlan, sekcja „A Foolish Consistency is the Hobgoblin of Little Minds”; tłum. własne

snake_case, spacje wokół = i nazwy bez polskich znaków

Komórka z sekcji 8. Oba warianty działają — różnią się tym, jak szybko je przeczytasz.

ŹLE

```
x=2
x= 2
imieNazwisko = "Jan Kowalski"  # camelCase – nie jest błędem, ale w Pythonie wolimy snake_case
```

DOBRZE

```
x = 2
imie_nazwisko = "Jan Kowalski"
```

imie_nazwisko

snake_case — Python

imieNazwisko

camelCase — np. Java, JavaScript

JESZCZE DWIE ZASADY Z NOTATNIKA

w projektach nazwy i komentarze po angielsku · bez polskich znaków w nazwach (imie, nie imię)

06

Zadania: test powie ci, czy działa

Ćw. 1 · Zadanie 0 z notatnika: Hello World!

5 min

1. Otwórz sekcję 9 i znajdź funkcję zadanie_0.
2. W miejsce pass wpisz jedno polecenie print.
3. Uruchom komórkę z funkcją, potem komórkę Sprawdź pod nią.

SEKCJA 9 — SZABLON (OPIS SKRÓCONY)

```
def zadanie_0():
    """Wydrukuj na standardowe wyjście: Hello Wo…"""
    # TODO: dopisz polecenie print poniżej
    pass
```

TEST NA PUSTYM SZABLONIE

❌ Oczekiwano 'Hello World!', otrzymano: ''

PYTANIE

Co wypisze test, jeśli zapomnisz wykrzyknika?

Rezultat ćw. 1: jedna linia print i test zaliczony

Rozwiązanie zadania 0 i wynik komórki Sprawdź z notatnika.

```python
def zadanie_0():
    """Wydrukuj na standardowe wyjście: Hello World! (dokładnie w tej postaci)."""
    print("Hello World!")
```

KOMÓRKA SPRAWDŹ

```
✅ Test zaliczony: poprawne 'Hello World!'
```

SPRAWDŹ U SIEBIE

- wielkie H i W
- wykrzyknik na końcu
- print, nie return — test czyta ekran

Ćw. 2 · Zadanie 1 z notatnika: funkcje geometryczne

12 min

1. W sekcji 10 uzupełnij pole_kwadratu: dla a < 0 podnieś ValueError, w przeciwnym razie zwróć pole.
2. Tak samo zrób obwod_kwadratu i pole_prostokata(a, b) — sprawdź oba boki.
3. Uruchom komórkę z funkcjami, potem komórkę Sprawdź.

SEKCJA 10 — PIERWSZA Z TRZECH (SKRÓCONA)

```python
def pole_kwadratu(a: float) -> float:
    """Zwróć pole kwadratu o boku a."""
    # TODO: zwróć właściwą wartość; dodaj walidację…
    raise NotImplementedError
```

WALIDACJA WEJŚCIA

```python
if a < 0:
    raise ValueError("...")
```

PYTANIE

Ile zwraca pole_prostokata(3, 4)? Co się stanie przy pole_kwadratu(-1)?

Rezultat ćw. 2: trzy funkcje, sześć asercji, zero błędów

Przykładowe rozwiązanie zadania 1 i wynik komórki Sprawdź.

```python
def pole_kwadratu(a: float) -> float:
    """Zwróć pole kwadratu o boku a."""
    if a < 0:
        raise ValueError("bok nie może być ujemny")
    return a * a

def obwod_kwadratu(a: float) -> float:
    """Zwróć obwód kwadratu o boku a."""
    if a < 0:
        raise ValueError("bok nie może być ujemny")
    return 4 * a

def pole_prostokata(a: float, b: float) -> float:
    """Zwróć pole prostokąta o bokach a i b."""
    if a < 0 or b < 0:
        raise ValueError("boki nie mogą być ujemne")
    return a * b
```

KOMÓRKA SPRAWDŹ

```
✅ Wszystkie testy zaliczone!
```

SPRAWDŹ U SIEBIE

- pole_kwadratu(5): 25
- obwod_kwadratu(5): 20
- pole_prostokata(3, 4): 12
- pole_kwadratu(-1): ValueError

Jeśli skończysz wcześniej

- Dopisz obwod_prostokata(a, b) z tą samą walidacją co pole_prostokata
- Dopisz pole_kola(r): użyj import math i math.pi
- Sformatuj swój kod narzędziem black albo sprawdź go narzędziem ruff

Propozycje z sekcji 11 notatnika.

Na następne zajęcia

- Dokończ zadania 0 i 1 — obie komórki Sprawdź mają pokazać ✅
- Przećwicz instrukcje warunkowe i pętle for oraz while
- Obejrzyj film podstawowy: youtube.com/watch?v=Hwckt4J96dI
- Dla chętnych: R. Hettinger, Beyond PEP 8, PyCon 2015 — youtube.com/watch?v=wf-BqAjZb8M

Linki z sekcji 11 notatnika.

Trzy rzeczy do zapamiętania

11.0

dzielenie / zawsze daje float

return

oddaje wartość dalej, print tylko pokazuje

snake_case

tak nazywasz zmienne i funkcje w Pythonie

DZIĘKUJĘ

dr inż. Szczepan Górtowski

Katedra Informatyki Ekonomicznej · Uniwersytet Ekonomiczny w Poznaniu

szczepan.gortowski@ue.poznan.pl · dyżury: kie.ue.poznan.pl/pl/dyzury/