

Python

Czwarty wykład

dr inż. Marcin Ciura, Uniwersytet Komisji Edukacji Narodowej

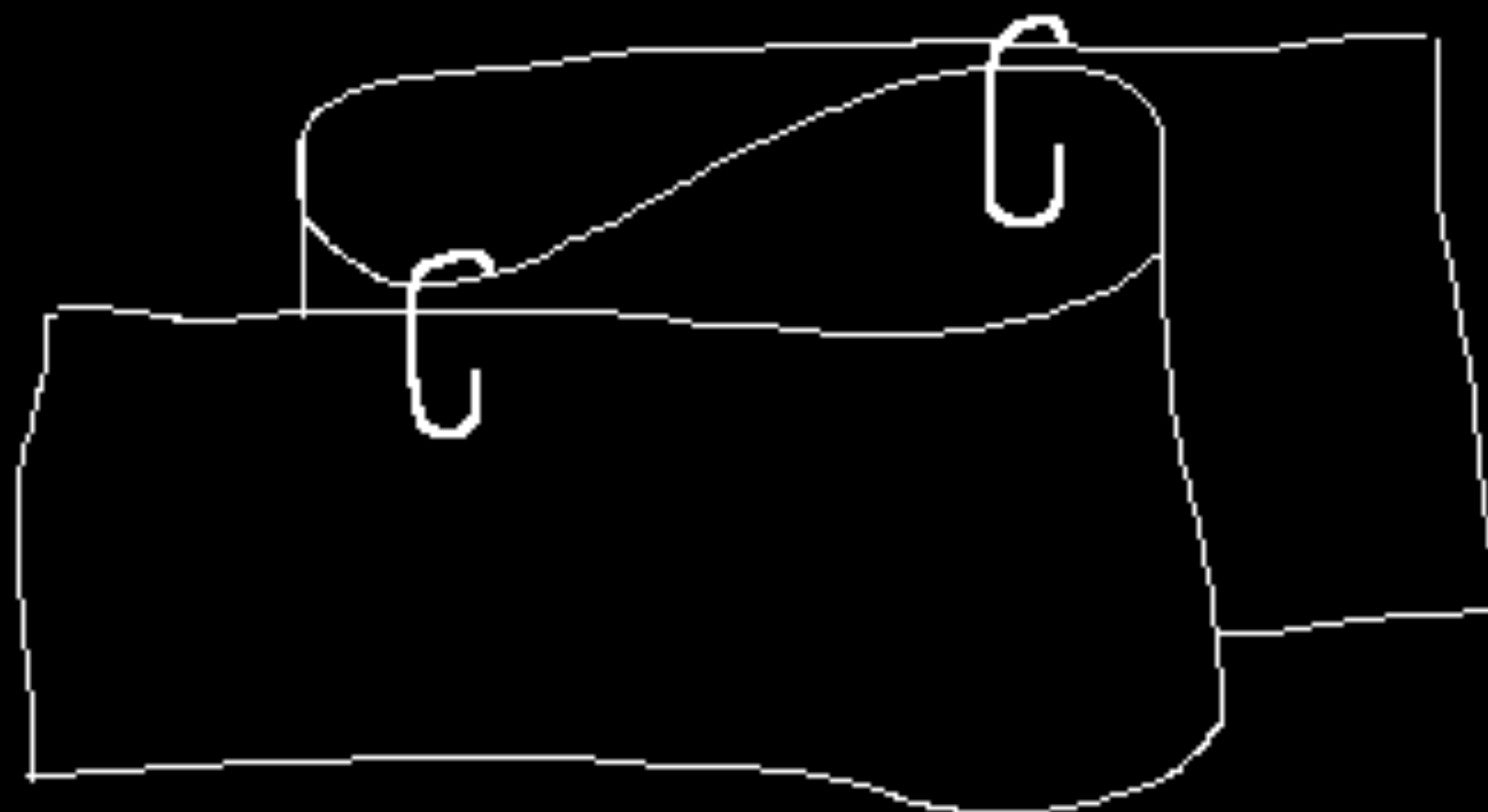
Przepraszam was za to, że obiecałem wam kilka sztuczek, a pokażę wam tylko jedną :-)

Sztuczka z banknotem
i dwoma spinaczami

Jeśli samodzielnie czytasz tę prezentację,
zachęcam cię: zrób tę sztuczkę
Wynik cię zaskoczy :-)

Sztuczka z banknotem i dwoma spinaczami (Bill Bowman, Seattle, 1954)

1. Zegnij banknot w literę S i zepnij go dwoma spinaczami



2. Śmiało pociągnij za końce banknotu. Nie podrze się :-)

Przepraszam was za to, że was oszukałem :-)
Dziś omówię więcej sztuczek
Proszę robić notatki :-)

Plan na dziś

+ Kilka sztuczek:

- łączenie operatorów porównania
- metoda `.get` słowników
- przypisanie krotki do krotki
- złożone operatory przypisania
- domyślne argumenty funkcji
- listy składane
- słowniki składane
- zbiory składane
- prawdziwe i nieprawdziwe obiekty

+ Krotki

+ Funkcje wbudowane:

- `sum`
- `sorted`
- `range`
- `enumerate`

+ Zbiory

+ Importowanie modułów

+ Wybrane funkcje czterech modułów:

- `math`
- `statistics`
- `random`
- `urllib`

Pierwsza sztuczka:
Łączenie operatorów porównania

Łączenie operatorów porównania

Zamiast pisać tak:

```
if x >= 90 and x <= 110:  
    print('Okolo 100')
```

możemy pisać tak:

```
if 90 <= x <= 110:  
    print('Okolo 100')
```

Lukier składniowy

Na takie sztuczki, które nie zmieniają istoty języka programowania, tylko dodają do niego wygodniejszy sposób zapisu, mówi się „lukier składniowy”

Jeśli ciastko bez lukru jest smaczne, to z lukrem będzie smaczniejsze. Byle nie przedobrzyć :-)

Jeśli język programowania jest wygodny, to z lukrem składniowym będzie wygodniejszy. Byle nie przedobrzyć

Druga sztuczka:
Metoda `.get` słowników

Metoda .get słowników

Jeśli `wyraz_pl` nie należy do słownika `słownik_pl_cz`, to wykonanie tej linii programu spowoduje wyjątek, czyli przerwie działanie programu:

```
wyraz_cz = słownik_pl_cz[wyraz_pl]
```

Metoda .get słowników

Dlatego bezpieczniej jest pisać tak:

```
if wyraz_pl in słownik_pl_cz:  
    wyraz_cz = słownik_pl_cz[wyraz_pl]  
else:  
    wyraz_cz = '???'
```

Metoda .get słowników

Dzięki metodzie .get możemy to zapisać krócej:

```
wyraz_cz = słownik_pl_cz.get(wyraz_pl, '???')
```

Metoda .get ma dwa argumenty:

- + klucz, którego wartość chcemy pobrać
- + wartość domyślna, którą zwróci ta metoda, jeśli klucza nie ma w słowniku

Metoda `.get` słowników — zagadka

Co robi taki fragment programu?

```
wyraz_cz = słownik_pl_cz.get(wyraz_pl, wyraz_pl)
```

Krotki

Krotki

Nazwa „krotka” (tuple) pochodzi od wyrazu „krotność”:

- + dwukrotny (double)
- + trzykrotny (triple)
- + czterokrotny (quadruple)
- + pięciokrotny (quintuple)
- + sześciokrotny (sextuple)
- + siedmiokrotny (septuple)

Krotki

Krotki są z pozoru podobne do list, tylko że:

- + listy otacza się nawiasami kwadratowymi, a krotki nawiasami okrągłymi ()
- + elementy list można zmieniać, a krotki są niezmiennie
- + elementy danej listy zazwyczaj są tego samego typu, a elementy danej krotki niekoniecznie

Krotki

```
współrzędne = (4, 3)
```

```
student = ('Jan', 'Nowak', 'bioinformatyka', 123456)
```

```
print(współrzędne[0]**2, współrzędne[1]**2)
```

```
print(student[1], student[0], student[2])
```

```
student[2] = 'biologia'
```

```
16 9
```

```
Nowak Jan bioinformatyka
```

```
TypeError: 'tuple' object does not support item assignment
```

Porównywanie krotek

Krotki porównuje się element po elemencie, od pierwszego elementu:

```
student = ('Jan', 'Kowalski', 'bioinformatyka', 123456)
print(student == ('Jan', 'Kowalski', 'bioinformatyka', 123456))
print(student == ('Jan', 'Kowalski', 'bioinformatyka', 123457))
print(student < ('Jan', 'Nowak', 'bioinformatyka', 654321))
```

True

False

True

Krotki

Łańcuchy, liczby i krotki mogą być kluczami słowników, bo są niezmiennie

Takie słowniki, które miałyby zmienne klucze, byłyby okropne: program mógłby wstawić do słownika wartość związaną z jakimś kluczem, a potem zmienić ten klucz i zgubić dostęp do wstawionej wartości

Trzecia sztuczka:
Przypisanie krotki do krotki

Przypisanie krotek do krotek

Oszukałem was: nawiasy okrągłe wokół krotek zwykle nie są potrzebne

Po prawej stronie operatora przypisania = zwykle się je pisze. Po lewej stronie operatora przypisania = zwykle się ich nie pisze

```
student = ('Jan', 'Kowalski', 'bioinformatyka', 123456)  
imię, nazwisko, kierunek, numer_indeksu = student
```

Przypisanie krotek do krotek

Zamiast pisać tak:

if $a > b$:

$tmp = a$ # tmp to skrót od "temporary variable", czyli „tymczasowa zmienna”

$a = b$

$b = tmp$ # Teraz na pewno $a \leq b$

możemy wygodnie zamieniać wartości zmiennych:

if $a > b$:

$a, b = b, a$ # Teraz na pewno $a \leq b$

Proszę mnie o coś zapytać :-)

Czwarta sztuczka:
Złożone operatory przypisania

Złożone operatory przypisania

Zamiast pisać tak:

```
x = x + 5
```

```
lista[i] = 2 * lista[i]
```

```
słownik[k] = słownik[k] - 1
```

możemy pisać tak:

```
x += 5
```

```
lista[i] *= 2
```

```
słownik[k] -= 1
```

To są złożone operatory
przypisania: += -= *= /=
%= **=

Operator reszty z dzielenia %

Operator % oblicza resztę z dzielenia lewej strony przez prawą:

```
print(10 % 3)
```

1

```
print(12 % 4)
```

0

```
if x % 2 == 0:
```

```
    print(f'{x} to liczba parzysta')
```

Funkcja wbudowana sum

Funkcja wbudowana sum

Funkcja **sum** sumuje elementy kolekcji liczb, czyli list, krotek, zbiorów, kluczy słowników

```
print(sum([4, 3.5, 2]))
```

9.5

Funkcja wbudowana `sorted`

Funkcja wbudowana sorted

Funkcja **sorted** sortuje kolekcje, czyli listy, krotki, zbiory, słowniki

Funkcja **sorted** zwraca nową, posortowaną listę

Funkcja **sorted** nie zmienia tej kolekcji, która jest jej argumentem

```
lista = sorted(kolekcja)
```


Funkcja wbudowana sorted

```
liczby = [3, 2, 6, 4]  
print(sorted(liczby))
```

```
[2, 3, 4, 6]
```

```
owoce = ['jabłko', 'gruszka', 'banan']  
print(sorted(owoce))
```

```
['banan', 'gruszka', 'jabłko']
```

Funkcja wbudowana `sorted`

Funkcja `sorted` z dodatkowym argumentem `reverse=True` sortuje kolekcje w kolejności malejącej

```
liczby = [3, 2, 6, 4]
```

```
print(sorted(liczby, reverse=True))
```

```
[6, 4, 3, 2]
```

Funkcja wbudowana range

Funkcja wbudowana range

Funkcja `range` generuje ciągi liczb całkowitych

Funkcja `range` jest często używana w pętlach `for`

Funkcja `range` ma trzy formy:

`range(stop)`

`range(start, stop)`

`range(start, stop, krok)`

Funkcja wbudowana range — wersja z jednym argumentem

`range(stop)` generuje kolejne liczby całkowite od 0 do `stop-1`

Funkcja `range` nigdy nie generuje wartości `stop`

```
for i in range(5):  
    print(i)
```

0
1
2
3
4

Funkcja wbudowana `range` — wersja z dwoma argumentami

`range(start, stop)` generuje liczby całkowite od `start` do `stop-1`
Funkcja `range` nigdy nie generuje wartości `stop`

```
for i in range(3, 7):  
    print(i)
```

3

4

5

6

Funkcja wbudowana range — wersja z trzema argumentami

`range(start, stop, krok)` pozwala określić krok, czyli wartość, o którą zwiększa się kolejna liczba ciągu. Funkcja `range` nigdy nie generuje wartości stop

```
for i in range(1, 10, 2):  
    print(i)
```

```
1  
3  
5  
7  
9
```

Funkcja wbudowana range — wersja z trzema argumentami

`range(start, stop, krok)` pozwala określić krok, czyli wartość, o którą zwiększa się kolejna liczba ciągu. Krok może być ujemny. Funkcja `range` nigdy nie generuje wartości stop

```
for i in range(5, 0, -1):  
    print(i)
```

```
5  
4  
3  
2  
1
```


Funkcja wbudowana *enumerate*

Funkcja wbudowana `enumerate`

Funkcja `enumerate` generuje krotki (indeks, wartość)

```
imiona = ['Anna', 'Bartek', 'Celina']
```

```
for indeks, imię in enumerate(imiona):  
    print(indeks, imię)
```

0 Anna

1 Bartek

2 Celina

Funkcja wbudowana `enumerate`

Argumentem funkcji `enumerate` może być dowolny obiekt, po którym można iterować: lista, krotka, łańcuch znaków, słownik lub zbiór

```
for indeks, znak in enumerate('Cześć'):  
    print(indeks, znak)
```

0 C

1 z

2 e

3 ś

4 ć

Proszę mnie o coś zapytać :-)

Piąta sztuczka:
Domyślne argumenty funkcji

Funkcja `print` — uzupełnienie

Kiedy funkcja `print` wypisze swój ostatni argument, domyślnie przechodzi do nowej linii

Kiedy podamy funkcji `print` argument `end`, funkcja `print` wypisze ten argument zamiast przejść do nowej linii

Funkcja print z argumentem end — przykład

```
print('Witaj,')  
print('świecie :-')
```

Witaj,
świecie

```
print('Witaj,', end=' ')  
print('świecie :-')
```

Witaj, świecie

Funkcja print z argumentem end — przykład

```
print('Liczby: ', end='')  
for i in range(1, 4):  
    print(i, end=' ')  
print('Gotowe!')
```

Liczby: 1 2 3 Gotowe!

Domyślne argumenty funkcji

Argument `reverse` funkcji `sorted` ma domyślną wartość
`False`

Argument `end` funkcji `print` ma domyślną wartość `'\n'`

Wartość domyślna to wartość, którą przyjmuje argument, gdy wywołujemy funkcję bez tego argumentu

Domyślne argumenty funkcji

Sami też możemy definiować funkcje z argumentami domyślnymi, przypisując im wartość w nagłówku funkcji za pomocą operatora =

Argumenty z wartościami domyślnymi muszą być na końcu listy argumentów

Domyślne argumenty funkcji

```
def powitaj(imię, wiadomość='Witaj'):  
    """Wypisuje wiadomość i imię."""  
    print(f'{wiadomość}, {imię}!')
```

```
powitaj('Jan')
```

Witaj, Jan!

```
powitaj('Anna', 'Cześć')
```

Cześć, Anna!

Zbiory

Zbiory

Zbiory to takie biedniejsze słowniki

Zbiory mają tylko klucze, nie mają wartości

Każdy klucz w zbiorze występuje jeden raz

Kolejność elementów w zbiorze nie ma znaczenia

```
liczby_pierwsze = {2, 3, 5, 7, 11, 13, 17, 19}
```

```
owoce = {'jabłko', 'gruszka', 'banan'}
```

Uwaga: {} to pusty słownik. Zbiór pusty tworzymy tak:

```
zbiór_pusty = set()
```

Działania na zbiorach

Czy element należy do zbioru:

```
if element in zbiór:  
    print('należy do zbioru')
```

```
if element not in zbiór:  
    print('nie należy do zbioru')
```

Po co nam są zbiory

Zamiast pisać tak:

```
if (littery == 'cz' or littery ==  
    'dz' or littery == 'dź' or  
    littery == 'dż' or littery ==  
    'rz' or littery == 'sz'):  
    print(f'{littery} to dwuznak')
```

możemy pisać tak:

```
DWUZNAKI = {  
    'cz', 'dz', 'dź', 'dż', 'rz', 'sz',  
}  
if littery in DWUZNAKI:  
    print(f'{littery} to dwuznak')
```

Zasada DRY

Proszę zwrócić uwagę na to, że dzięki zbiorom nie powtarzamy sześć razy fragmentu `lityr ==`

To przykład zastosowania zasady

DRY

(Don't Repeat Yourself)

Działania na zbiorach

Suma dwóch zbiorów ($a \mid b$) zawiera te elementy, które należą co najmniej do jednego z tych zbiorów:

```
małe_liczby = {1, 2, 3, 4}
```

```
liczby_pierwsze = {2, 3, 5, 7, 11, 13, 17, 19}
```

```
suma_zbiorów = małe_liczby | liczby_pierwsze
```

```
print(suma_zbiorów)
```

```
{1, 2, 3, 4, 5, 7, 11, 13, 17, 19}
```

Działania na zbiorach

Część wspólna dwóch zbiorów ($a \& b$) zawiera te elementy, które należą do obu tych zbiorów naraz:

```
małe_liczby = {1, 2, 3, 4}
liczby_pierwsze = {2, 3, 5, 7, 11, 13, 17, 19}
część_wspólna_zbiorów = małe_liczby & liczby_pierwsze
print(część_wspólna_zbiorów)
{2, 3}
```

Działania na zbiorach

Różnica dwóch zbiorów ($a - b$) zawiera te elementy, które należą do pierwszego zbioru i nie należą do drugiego zbioru:

```
małe_liczby = {1, 2, 3, 4}
liczby_pierwsze = {2, 3, 5, 7, 11, 13, 17, 19}
różnica_zbiorów = małe_liczby - liczby_pierwsze
print(różnica_zbiorów)
{1, 4}
```

Zbiory — podsumowanie

```
zbiór = {elem1, elem2}
zbiór_pusty = set()

if element in zbiór:
    print('Należy')
if element not in zbiór:
    print('Nie należy')
```

```
suma_zbiorów = zb1 | zb2
część_wspólna = zb1 & zb2
różnica_zbiorów = zb1 - zb2
```

Szósta sztuczka:
Listy składane

Listy składane

Dzięki listom składanym możemy tworzyć listy na podstawie innych kolekcji w jednej linii kodu

Proste listy składane zapisujemy tak:

[wyrażenie for element in kolekcja]

Listy składane

Zamiast pisać tak:

```
sześciiany = []  
for i in range(10):  
    sześciiany.append(i**3)
```

możemy pisać tak:

```
sześciiany = [i**3 for i in range(10)]
```

Listy składane z warunkiem

Listy składane z warunkiem zapisujemy tak:

[wyrażenie for element in kolekcja if warunek]

Listy składane

Stwórz listę tych wyrazów, które mają więcej niż 5 liter:

```
wyrazy = ['jabłko', 'kiwi', 'banan', 'arbuz', 'ananas']
```

```
długie_wyrazy = [s for s in wyrazy if len(s) > 5]
```

```
print(długie_wyrazy)
```

```
['jabłko', 'ananas']
```

Listy składane

Możemy zagnieżdżać pętle w listach składanych

```
[wyrażenie for mniejsza_kolekcja in kolekcja  
           for element in mniejsza_kolekcja  
           if warunek]
```

```
[wyrażenie for element1 in kolekcja1  
           for element2 in kolekcja2  
           if warunek]
```

Listy składane

Wybierz z macierzy liczby parzyste:

```
macierz = [  
    [ 6, 3, 7 ],  
    [ 2, 9, 4 ],  
    [ 5, 0, 9 ],  
]
```

```
parzyste = [x for wiersz in macierz for x in wiersz if x % 2 == 0]
```

```
print(parzyste)
```

```
[6 2 4 0]
```

Zbiory składowane

Zbiory składane

Zbiory składane tworzymy tak samo, jak listy składane, tylko otaczamy je nawiasami klamrowymi { }

{wyrażenie for element in kolekcja}

{wyrażenie for element in kolekcja if warunek}

{wyrażenie for mniejsza_kolekcja in kolekcja
 for element in mniejsza_kolekcja
 if warunek}

Zbiory składane

Wypisz listę wyrazów bez powtórzeń, posortowaną alfabetycznie:

```
tekst = 'Padał deszcz, padał na dach, padał na okna'  
print(sorted({wyraz.lower() for wyraz in tekst.split()}))  
['dach,', 'deszcz,', 'na', 'okna', 'padał']
```

Słowniki składowe

Słowniki składane

Słowniki składane tworzymy podobnie do list składanych i zbiorów składanych

{klucz: wartość for element in kolekcja}

{klucz: wartość for element in kolekcja if warunek}

{klucz: wartość for mniejsza_kolekcja in kolekcja
for element in mniejsza_kolekcja
if warunek}

Słowniki składane

Stwórz słownik czesko-polski ze słownika polsko-czeskiego:

```
słownik_cz_pl = {cz: pl for pl, cz in słownik_pl_cz.items() }
```

Słowniki składane

Stwórz słownik, w którym kluczami są wyrazy,
a wartościami długości tych wyrazów:

```
wyrazy = ['jabłko', 'kiwi', 'banan', 'arbuz', 'ananas']  
długości_wyrazów = {w: len(w) for w in wyrazy}  
print(długości_wyrazów)
```

```
{'jabłko': 6, 'kiwi': 4, 'banan': 5, 'arbuz': 5, 'ananas': 6}
```

Proszę mnie o coś zapytać :-)

Siódma sztuczka:
Prawdziwe i nieprawdziwe obiekty

Prawdziwe i nieprawdziwe obiekty

Każdy obiekt w Pythonie ma wartość logiczną, której możemy użyć bezpośrednio w instrukcji warunkowej if:

if obiekt:

```
print('Obiekt jest prawdziwy (truthy)')
```

if not obiekt:

```
print('Obiekt jest nieprawdziwy (falsy)')
```

Nieprawdziwe obiekty

False — jedna z dwóch wartości logicznych

0 i 0.0 — zero całkowite i rzeczywiste

" lub "" — pusty łańcuch znaków

[] — pusta lista

() — pusta krotka

{} — pusty słownik

set() — zbiór pusty

None — specjalny obiekt, który reprezentuje brak wartości

Prawdziwe obiekty

Wszystkie inne obiekty, na przykład:

True — jedna z dwóch wartości logicznych

1 lub **2.5** — niezerowe liczby całkowite i rzeczywiste

'ser' lub **"szynka"** — niepuste łańcuchy znaków

['mielonka', 'szynka'] — niepuste listy

(3, 5) — niepuste krotki

{'ser': 4.99, 'szynka': 49.99} — niepuste słowniki

{'Ala', 'As'} — niepuste zbiory

Prawdziwe obiekty

Zamiast pisać tak:

```
lista = []  
if lista != []:  
    print('Lista nie jest pusta')  
else:  
    print('Lista jest pusta')
```

albo tak:

```
if len(lista) > 0:  
    print('Lista nie jest pusta')  
else:  
    print('Lista jest pusta')
```

można pisać tak, jak na
następnym slajdzie

Prawdziwe obiekty

Przykład:

```
lista = []  
if lista:  
    print('Lista nie jest pusta')  
else:  
    print('Lista jest pusta')
```

albo*:

```
if not lista:  
    print('Lista jest pusta')  
else:  
    print('Lista nie jest pusta')
```

* Radzę unikać not
w warunkach, jeśli to możliwe

Importowanie modułów

Importowanie modułów

Takie funkcje, które przydają się w wielu programach, zapisuje się w modułach

Moduł to po prostu plik tekstowy z rozszerzeniem **.py**

Aby używać funkcji z jakiegoś modułu, należy zaimportować ten moduł

Importowanie modułów

Dobry zwyczaj: importowanie modułów na początku innych modułów

Tak się importuje moduł:

import nazwa_modułu

Tak się potem wywołuje funkcje z tego modułu:

nazwa_modułu.nazwa_funkcji(argumenty_funkcji)

Importowanie modułów

Tak można importować moduł, który ma długą nazwę:

```
import długa_nazwa_modułu as nazwa
```

Tak się potem wywołuje funkcje z tego modułu:

```
nazwa.nazwa_funkcji(argumenty_funkcji)
```

Importowanie modułów

Tak można importować z modułu tylko wybrane funkcje:

from nazwa_modułu import funkcja1, funkcja2, funkcja3

Potem wywołujemy te funkcje tak, jakby były zdefiniowane w tym module, w którym są wywoływane

Nie podajemy nazwy modułu, w którym są zdefiniowane

funkcja1(argumenty_funkcji)

Importowanie modułów

Tak można zaśmiecić „przestrzeń nazw” nie wiadomo czym. Proszę tak nie robić:

from nazwa_modułu import *

Biblioteka standardowa

Biblioteka standardowa

Python szuka modułów w określonej kolejności:

- + najpierw sprawdza folder, z którego uruchomiono program
- + potem sprawdza foldery ze zmiennej `sys.path`

Foldery ze zmiennej `sys.path` zawierają bibliotekę standardową i być może biblioteki zewnętrzne

Niektóre moduły biblioteki standardowej

Biblioteka standardowa jest zawsze zainstalowana razem z Pythonem

Biblioteki zewnętrzne trzeba instalować osobno

W bibliotece standardowej Pythona jest około 160 modułów. Omówię kilka funkcji z 4 modułów

Modul math

Moduł math

Moduł `math` zawiera popularne funkcje i stałe matematyczne

```
import math  
print(math.pi)
```

3.141592653589793

Moduł math

Funkcja `math.log(x, podstawa=e)` zwraca logarytm x o danej podstawie. Jeśli podstawa nie jest podana, używa podstawy $e=2.718281828459045\dots^*$, czyli zwraca logarytm naturalny

* e z dokładnością do 15 miejsc po przecinku równa się 2 przecinek 7, potem dwa razy rok urodzenia Lwa Tołstoja, czyli 1828, a potem kąty równoramienne trójkąta prostokątnego: 45, 90, 45

Moduł math

Funkcje `math.sin(x)`, `math.cos(x)` i `math.tan(x)`* zwracają sinus, cosinus i tangens x

Uwaga: x musi być podane w mierze łukowej, czyli w radianach, nie w stopniach

* Niektórzy myślą że język symboli matematycznych jest uniwersalny. To prawda, ale są drobne różnice między narodami. Anglosasi skracają tangens do „tan”, a my do „tg”. Ale to jeszcze nic: Włosi nazywają sinus „seno”, więc skracają go do „sen” :-)

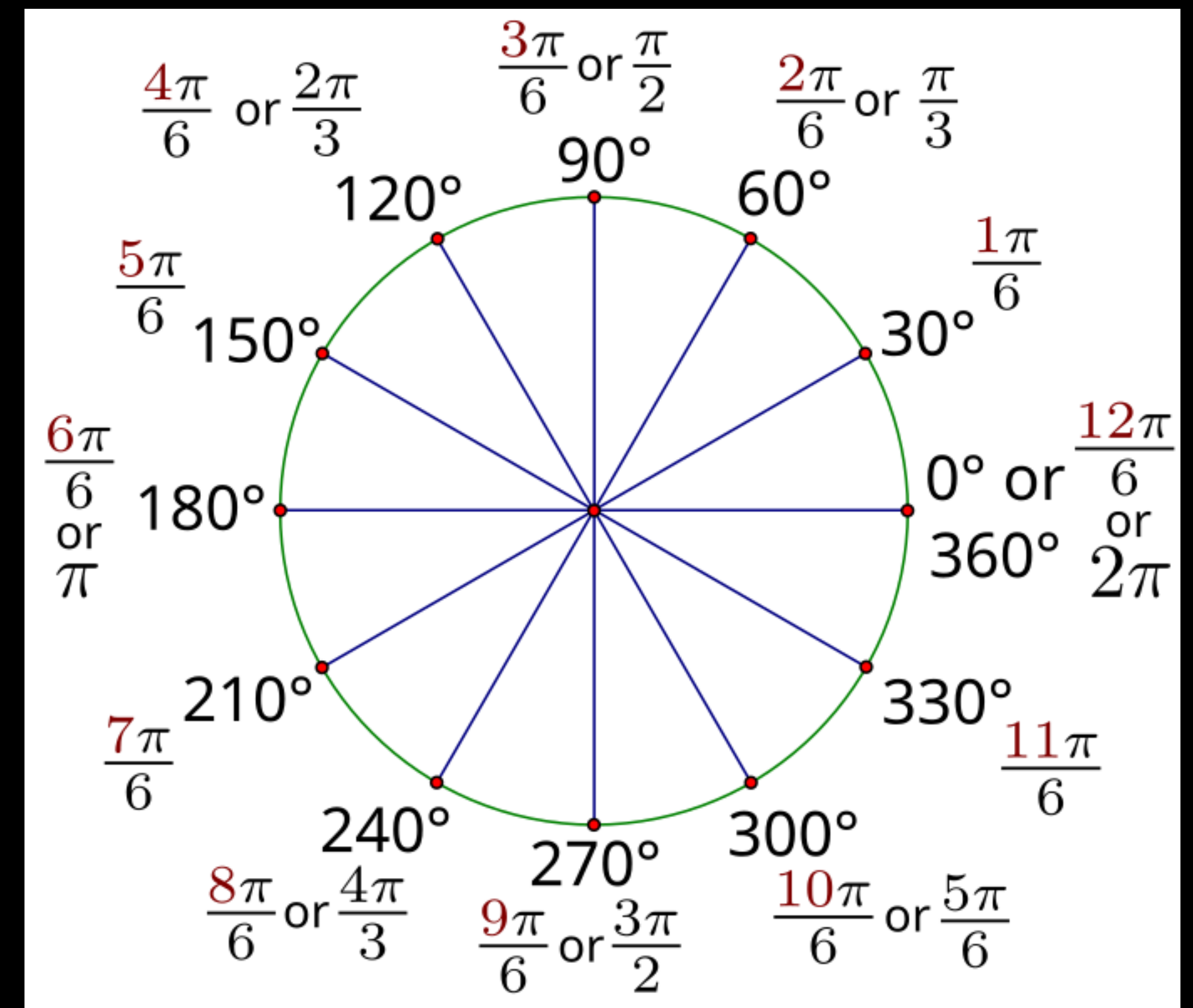
Moduł math

Aby zamienić kąt wyrażony
w stopniach na radiany,
należy użyć funkcji
`math.radians(kąt)`

```
import math
```

```
print(math.radians(60))
```

```
1.0471975511965976
```



Moduł math

Funkcja `math.sqrt` (od „square root”) oblicza pierwiastek kwadratowy, co daje ten sam wynik, co podniesienie liczby do potęgi 0.5:

```
import math  
print(math.sqrt(9))  
print(9 ** 0.5)  
3.0  
3.0
```


Modul statistics

Moduł statistics

Moduł `statistics` zawiera podstawowe funkcje statystyczne
Funkcja `statistics.mean` oblicza średnią arytmetyczną listy liczb

```
import statistics  
ceny = [2, 5, 4, 4]  
print(statistics.mean(lista))  
print(sum(lista) / len(lista))  
3.75  
3.75
```

Moduł statistics

Funkcja `statistics.median` oblicza medianę listy liczb.
Ta lista nie musi być posortowana

```
import statistics
```

```
print(statistics.median([1, 2, 4, 8, 3]))
```

3

Modul random

Moduł random

Moduł `random` służy do generowania losowych liczb

Funkcja `random.randint(a, b)` (od „random integer”) zwraca przy każdym wywołaniu losową liczbę całkowitą między `a` i `b` włącznie

Tak się symuluje rzut kostką do gry:

```
import random
```

```
print(random.randint(1, 6), random.randint(1, 6), random.randint(1, 6))
```

```
4 5 1
```

Moduł random

Funkcja `random.choice(kolekcja)` zwraca przy każdym wywołaniu losowy element kolekcji. Nie zmienia kolekcji

```
import random  
owoce = ['jabłko', 'gruszka', 'banan']  
print(random.choice(owoce), random.choice(owoce),  
       random.choice(owoce))  
banan gruszka banan
```

Modul urllib

Moduł urllib

Pakiet modułów `urllib` (od „URL library”; URL to Uniform Resource Locator, czyli adres internetowy) służy do odczytywania danych z Internetu
Najważniejszy moduł tego pakietu to `urllib.request`

Moduł urllib

Szablon pobierania danych z Internetu:

```
import urllib.request  
with urllib.request.urlopen('https://www.example.com') as response:  
    print(response.read().decode('utf-8'))
```

To wszystko na dziś
Proszę mi zadać pytanie :-)

Czy każdy z was zrozumiał
wszystko z tego wykładu?

Do zobaczenia na następnym wykładzie
o wyrażeniach regularnych
Gdybym był genetykiem, z całego Pythona zapamiętałbym
tylko wyrażenia regularne :-)

Źródła zdjęć

[https://commons.wikimedia.org/wiki/
File:30 degree rotations expressed in radian measure
.svg](https://commons.wikimedia.org/wiki/File:30_degree_rotations_expressed_in_radian_measure.svg)