PYTHON

Powtórzenie

Korzystanie z pomocy w IDLE

1. **Otwarcie dokumentacji**:
   - W menu głównym kliknij **Help** → **Python Docs**.
   - Otworzy się okno z oficjalną dokumentacją Pythona.
2. **Funkcja `help()`**:
   - Wpisz `help()` w interaktywnym oknie IDLE.
   - Możesz uzyskać pomoc dotyczącą konkretnej funkcji lub modułu:

```
help(math)      # Pomoc dotycząca modułu math
help(math.lcm)  # Pomoc dotycząca funkcji lcm
help(str)
help(str.upper())
```

3. **Korzystanie z `dir()`**:
   - Aby zobaczyć dostępne metody i atrybuty dla obiektu:

```
dir(math)  # Wyświetla listę funkcji w module math
```

4. **Podpowiedzi kontekstowe**:
   - `Ctrl+Space` - IDLE oferuje podpowiedzi dotyczące składni, funkcji i parametrów po wpisaniu części nazwy funkcji.

Funkcje w module `random`

1. `random()`

- **Opis**: Generuje liczbę zmiennoprzecinkową z przedziału [0.0, 1.0).
- **Przykład**:

```
import random
result = random.random()  # Output: np. 0.654321
```

2. `randint(a, b)`

- **Opis**: Zwraca losową liczbę całkowitą z zakresu [a, b], włącznie z obu stron.
- **Przykład**:

```
result = random.randint(1, 10)  # Output: np. 7
```

3. `choice(seq)`

- **Opis**: Wybiera losowy element z niepustej sekwencji (np. lista, ciąg).
- **Przykład**:

```
result = random.choice(['a', 'b', 'c'])  # Output: np. 'b'
```

4. `shuffle(seq)`

- **Opis**: Tasuje listę w miejscu.
- **Przykład**:

```
items = [1, 2, 3]
random.shuffle(items)  # Output: np. [3, 1, 2]
```

5. `sample(population, k)`

- **Opis**: Zwraca listę `k` unikalnych elementów losowo wybranych z `population`.
- **Przykład**:

```
result = random.sample(range(1, 10), 3)  # Output: np. [4, 7, 1]
```

6. `uniform(a, b)`

- **Opis**: Generuje losową liczbę zmiennoprzecinkową z zakresu [a, b].
- **Przykład**:

```
result = random.uniform(1.5, 5.5)  # Output: np. 3.789
```

7. `randrange(start, stop[, step])`

- **Opis**: Zwraca losową liczbę z zakresu `start` do `stop` (bez `stop`), z opcjonalnym krokiem `step`.
- **Przykład**:

```
result = random.randrange(0, 10, 2)  # Output: np. 4
```

8. Pomoc random

- **Przykład**:

```
import random
dir(random)
help(random.choice)
```

Funkcje `min()`, `max()`, `sum()`

1. `min()`

Funkcja `min()` zwraca najmniejszą wartość z podanych elementów lub listy.

Przykłady:

```
print(min([3, 5, 7, 2, 8]))  # 2
print(min(3, 5, 7, 2, 8))    # 2
```

2. `max()`

Funkcja `max()` zwraca największą wartość z podanych elementów lub listy.

Przykłady:

```
print(max([3, 5, 7, 2, 8]))  # 8
print(max(3, 5, 7, 2, 8))    # 8
```

3. `sum()`

Funkcja `sum()` oblicza sumę elementów w liście lub krotce.

Przykłady:

```
print(sum([3, 5, 7, 2, 8]))  # 25
print(sum((3, 5, 7, 2, 8)))  # 25
```

`min()` i `max()` ze stringami

Działanie:

- `min()`: Zwraca najmniejszy (leksykograficznie) znak lub słowo w stringu.
- `max()`: Zwraca największy (leksykograficznie) znak lub słowo w stringu.

Przykłady:

`min()` ze stringiem

```
text = "hello"
print(min(text))  # Output: 'e' (najmniejszy znak w kolejności ASCII)
```

`max()` ze stringiem

```
text = "hello"
print(max(text))  # Output: 'o' (największy znak w kolejności ASCII)
```

`min()` i `max()` z listą słów

```
words = ["apple", "banana", "cherry"]
print(min(words))  # Output: 'apple' (pierwsze w kolejności leksykograficznej)
print(max(words))  # Output: 'cherry' (ostatnie w kolejności leksykograficznej)
```

Zasada:

- `min()` i `max()` działają na zasadzie porównania leksykograficznego, gdzie porównywane są kody ASCII znaków.

`range(start, stop, step)`

Opis:

- Generuje sekwencję liczb całkowitych od `start` (włącznie) do `stop` (wyłącznie) z krokiem `step`.
- Parametry:
  - `start` (opcjonalny): Początek sekwencji. Domyślnie `0`.
  - `stop`: Koniec sekwencji (niewłączany).
  - `step` (opcjonalny): Odstęp między kolejnymi liczbami. Domyślnie `1`.

Przykłady:

```
range(5)            # Output: 0, 1, 2, 3, 4
range(1, 5)         # Output: 1, 2, 3, 4
range(1, 10, 2)     # Output: 1, 3, 5, 7, 9
range(10, 0, -2)    # Output: 10, 8, 6, 4, 2
```

Slices `list[start:stop:step]`

Opis:

- Umożliwia wycięcie fragmentu sekwencji (np. listy, ciągu) od indeksu `start` do `stop` z krokiem `step`.
- Parametry:
  - `start` (opcjonalny): Indeks początkowy (włącznie). Domyślnie `0`.
  - `stop`: Indeks końcowy (niewłączany).
  - `step` (opcjonalny): Odstęp między elementami. Domyślnie `1`.

Przykłady:

```
lst = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

lst[2:5]            # Output: [2, 3, 4]
lst[:5]             # Output: [0, 1, 2, 3, 4]
lst[::2]            # Output: [0, 2, 4, 6, 8]
lst[1:8:2]          # Output: [1, 3, 5, 7]
lst[::-1]           # Output: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Slices ze stringami w Pythonie

Opis:

- Slices działają podobnie jak w przypadku list, ale tutaj manipulujemy znakami w ciągu znaków (stringu).
- Format: `string[start:stop:step]`
  - `start` (opcjonalny): Indeks początkowy (włącznie).
  - `stop`: Indeks końcowy (niewłączany).
  - `step` (opcjonalny): Odstęp między znakami.

Przykłady:

Indeksy dla `"Hello, World!"` (poziomo)

Znak	H	e	l	l	o	,	(spacja)	W	o	r	l	d	!
Dodatni	0	1	2	3	4	5	6	7	8	9	10	11	12
Ujemny	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
text = "Hello, World!"

# 1. Wycinanie fragmentu
print(text[0:5])     # Output: 'Hello'

# 2. Pomijanie początku (od początku do `stop`)
print(text[:5])      # Output: 'Hello'

# 3. Pomijanie końca (od `start` do końca)
print(text[7:])      # Output: 'World!'

# 4. Użycie kroku
print(text[::2])     # Output: 'Hlo ol!'
```

```
# 5. Odwracanie stringa
print(text[::-1])    # Output: '!dlroW ,olleH'

# 6. Wycinanie z ujemnymi indeksami
print(text[-6:-1])   # Output: 'World'
```

Uwagi:

- **Ujemne indeksy**: Liczone od końca stringa (`-1` to ostatni znak).
- **Niepodanie `start` lub `stop`**: Domyślnie zaczyna od początku lub kończy na końcu stringa.

Działania

```
print(3 + 2)
print(3 - 2)
print(3 * 2)
print(3 / 2.0) #dzielenie
print(3 // 2) #dzielenie całkowite
print(3 % 2) #reszta z dzielenia
print(3 ** 2) #potęgowanie
```

```
from math import *

# Pierwiastek kwadratowy
print(sqrt(2))  # Output: 1.4142135623730951

# Silnia
print(factorial(4))  # Output: 24

# Podłoga - największa liczba całkowita, która jest równa lub mniejsza od danej
print(floor(2.3))  # Output: 2

# Sufit - najmniejsza liczba całkowita, która jest równa lub większa od danej
print(ceil(2.3))  # Output: 3

# Liczba Pi
print(pi)  # Output: 3.141592653589793

# Potęgowanie
print(pow(2, 3))  # Output: 8  (2^3)

# Najmniejsza wspólna wielokrotność
print(lcm(12, 18))  # Output: 36

# Największy wspólny dzielnik
print(gcd(12, 18))  # Output: 6

# Zaokrąglanie liczby
print(round(2.345))       # Output: 2
print(round(2.345, 2))    # Output: 2.35
print(round(2.675, 2))    # Output: 2.67  # Efekt zaokrąglania liczby binarnej
```

Operacje na listach w Pythonie

1. append()

- Dodaje element na końcu listy.

```
my_list = [1, 2, 3]
my_list.append(4)
print(my_list)  # [1, 2, 3, 4]
```

2. clear()

- Usuwa wszystkie elementy z listy.

```
my_list = [1, 2, 3]
my_list.clear()
print(my_list)  # []
```

3. count()

- Zwraca liczbę wystąpień elementu w liście.

```
my_list = [1, 2, 2, 3, 2]
print(my_list.count(2))  # 3
```

4. extend()

- Dodaje elementy z innej listy na końcu listy.

```
my_list = [1, 2]
my_list.extend([3, 4])
print(my_list)  # [1, 2, 3, 4]
```

5. index()

- Zwraca indeks pierwszego wystąpienia elementu w liście.

```
my_list = [1, 2, 3]
print(my_list.index(2))  # 1
```

6. insert()

- Wstawia element na określoną pozycję w liście.

```
my_list = [1, 2, 3]
my_list.insert(1, 'a')
print(my_list)  # [1, 'a', 2, 3]
```

7. remove()

- Usuwa pierwsze wystąpienie elementu z listy.
- Jeśli element nie istnieje w liście, podnosi wyjątek `ValueError`.

```
my_list = [1, 2, 3, 2]
my_list.remove(2)
print(my_list)  # [1, 3, 2]
```

8. reverse()

- Odwraca kolejność elementów w liście.

```
my_list = [1, 2, 3]
my_list.reverse()
print(my_list)  # [3, 2, 1]
```

9. reversed() i `[::-1]`

- `reversed()`: Zwraca odwrócony iterator.
- `[::-1]`: Odwraca sekwencję.

Przykład `reversed()`:

```
numbers = [1, 2, 3, 4, 5]
reversed_numbers = list(reversed(numbers))
print(reversed_numbers)  # Output: [5, 4, 3, 2, 1]
```

Przykład `[::-1]`:

```
text = "hello"
reversed_text = text[::-1]
print(reversed_text)  # Output: "olleh"
```

10. sort()

- Sortuje listę w porządku rosnącym (lub malejącym, jeśli podasz argument `reverse=True`).

```
my_list = [3, 1, 2]
my_list.sort()
print(my_list)  # [1, 2, 3]
```

11. sorted()

- Zwraca posortowaną listę elementów, nie modyfikując oryginalnej listy.

```
numbers = [3, 1, 4, 1, 5, 9]
sorted_numbers = sorted(numbers)
print(sorted_numbers)  # Output: [1, 1, 3, 4, 5, 9]

# Przykład z ciągiem znaków
text = "python"
sorted_text = sorted(text)
print(sorted_text)  # Output: ['h', 'n', 'o', 'p', 't', 'y']
```

12. find()

- Zwraca indeks pierwszego wystąpienia elementu w liście (zwraca `-1`, jeśli element nie występuje).

```
my_list = [1, 2, 3, 4, 5]
print(my_list.find(3))  # Output: 2
print(my_list.find(6))  # Output: -1
```

Turtle

```
tracer(0)
trojkat(40, "1")
update()
done()
```

```
tracer(0)
trojkat(40, "1")
update()
mainloop()
```

Funkcje w module `turtle`

1. setx(x)

- Ustawia współrzędną x pozycji żółwia.

```
import turtle
turtle.setx(100)  # Ustawienie pozycji żółwia na (100, 0)
```

2. sety(y)

- Ustawia współrzędną y pozycji żółwia.

```
import turtle
turtle.sety(100)  # Ustawienie pozycji żółwia na (0, 100)
```

3. pensize(width)

- Ustala grubość pióra (linii rysowanej przez żółwia).

```
import turtle
turtle.pensize(5)  # Grubość linii 5 pikseli
```

4. speed(speed)

- Ustawia prędkość rysowania żółwia. Może przyjmować wartości od `0` (najszybsza) do `10` (najwolniejsza), lub specjalne wartości takie jak `fastest`, `fast`, `normal`, `slow`, `slowest`.

```
import turtle
turtle.speed(10)  # Ustawienie najwolniejszej prędkości
```

5. goto(x, y)

- Przenosi żółwia do podanych współrzędnych `(x, y)` na ekranie.

```
import turtle
turtle.goto(100, 100)  # Przenosi żółwia do punktu (100, 100)
```

6. fillcolor(color)

- Ustawia kolor wypełnienia figury (np. koła) rysowanej przez żółwia.

```
import turtle
turtle.fillcolor("red")  # Kolor wypełnienia to czerwony
```

7. begin_fill()

- Rozpoczyna wypełnianie figury.

```
import turtle
turtle.begin_fill()  # Rozpoczyna wypełnianie
```

8. end_fill()

- Kończy wypełnianie figury. Cała figura, która została narysowana od `begin_fill()` do `end_fill()`, zostanie wypełniona ustawionym kolorem.

```
import turtle
turtle.end_fill()  # Kończy wypełnianie figury
```

9. clear()

- Czyści ekran, usuwając wszystkie narysowane elementy, ale nie resetuje pozycji żółwia.

```
import turtle
turtle.clear()  # Usuwa wszystkie narysowane elementy
```

10. reset()

- Czyści ekran i resetuje wszystkie ustawienia, w tym pozycję, orientację i inne parametry żółwia.

```
import turtle
turtle.reset()  # Resetuje ustawienia żółwia
```

11. position()

- Zwraca bieżącą pozycję żółwia jako krotkę `(x, y)`.

```
import turtle
print(turtle.position())  # Zwraca aktualną pozycję żółwia, np. (100.00, 200.00)
```

12. done()

- Zakończenie rysowania. Używane do zakończenia programu i pozostawienia okna otwartego.

```
import turtle
turtle.done()  # Zakończenie programu, okno pozostaje otwarte
```

13. bye()

- Zamyka okno rysowania żółwia.

```
import turtle
turtle.bye()  # Zamknięcie okna
```

14. mainloop()

- Rozpoczyna główną pętlę programu. Używane do utrzymania okna otwartego w trybie graficznym (okno nie zamknie się automatycznie).

```
import turtle
turtle.mainloop()  # Uruchamia główną pętlę programu
```