ЖИТОМИРСЬКА ПОЛІТЕХНІКА
ДЕРЖАВНИЙ УНІВЕРСИТЕТ

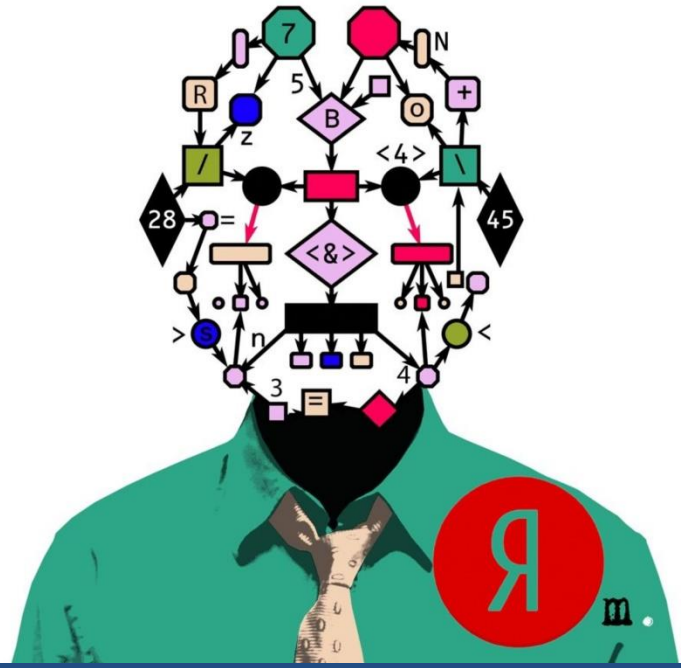
@ZTUEDUUA

АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРИЗОВАНИХ ІНФОРМАЦІЙНО-ВИМІРЮВАЛЬНИХ СИСТЕМ

Лекція 4

Тема:Використання керуючих конструкцій мови Python. Реалізація розгалужень

1. Оператори порівняння Python.
2. Види операторів порівняння.
3. Умовний оператор if. Імітація оператор вибору switch/case в мові Python.



1.Оператори порівняння Python

Оператори порівняння є бінарні, тобто для обчислення потребують двох операндів. Результатом будь-якого оператора порівняння є логічне значення True або False. Значення True встановлюється, якщо операція порівняння справджується (істина, правда).

Умовні вирази

Ряд операцій представляють умовні висловлювання. Всі ці операції приймають два операнди і повертають логічне значення, яке в Python представляє тип bool. Існує тільки два логічні значення - True (вираз істинно) і False (вираз хибно).

Операції порівняння:

- ==

  Повертає True, якщо обидва операнди рівні. Інакше повертає False.

- !=

  Повертає True, якщо обидва операнди НЕ рівні. Інакше повертає False.

- > (більше ніж)

  Повертає True, якщо перший операнд більший за другий.

- < (менше ніж)

  Повертає True, якщо перший операнд менший за другий.

- >= (більше або дорівнює)

  Повертає True, якщо перший операнд більше або дорівнює другому.

- <= (менше або дорівнює)

  Повертає True, якщо перший операнд менший або дорівнює другому.

2.Види операторів порівняння

Приклад операції порівняння:

```
a = 5
b = 6
result = 5 == 6  # сохраняем результат операции в переменную
print(result)  # False - 5 не равно 6
print(a != b)  # True
print(a > b)  # False - 5 меньше 6
print(a < b)  # True

bool1 = True
bool2 = False
print(bool1 == bool2)  # False - bool1 не равно bool2
```

Логічні операції:

Оператор **and** (логическое умножение) применяется к двум операндам:

```
x and y
```

Спочатку оператор **and** оцінює вираз x, і якщо він дорівнює **False**, то повертається його значення. Якщо воно дорівнює True, то оцінюється другий операнд – y і повертається значення y.

```
age = 22
weight = 58
result = age > 21 and weight == 58
print(result)  # True
```

В даному випадку оператор **and** порівнює результати двох виразів: **age > 21 weight == 58**. І якщо обидва ці вирази повертають **True**, то оператор **and** також повертає **True** (формально повертається значення останнього операнда).

```
result = 4 and "w"
print(result)  # w, так как 4 равно True, поэтому возвращается значение последнего операнда

result = 0 and "w"
print(result)  # 0, так как 0 эквивалентно False
```

У даному випадку число 0 і порожній рядок "" розглядаються як False, всі інші числа та непусті рядки еквівалентні **True**

- **or** (логічне додавання) також застосовується до двох операндів:

```
x or y
```

Спочатку оператор **or** оцінює вираз x, і якщо він дорівнює **True**, то повертається його значення. Якщо воно дорівнює **False**, то оцінюється другий операнд – y і повертається значення y. Наприклад:

```
age = 22
isMarried = False
result = age > 21 or isMarried
print(result)  # True, так как выражение age > 21 равно True
```

- **not** (логічне заперечення)

  Повертає **True**, якщо вираз дорівнює **False**

```
age = 22
isMarried = False
print(not age > 21)  # False
print(not isMarried)  # True
print(not 4)  # False
print(not 0)  # True
```

Оператор in

Оператор **in** повертає **True**, якщо в деякому наборі значень є певне значення. Він має таку форму:

```
значение in набор_значений
```

Наприклад, рядок представляє набір символів. І за допомогою оператора **in** ми можемо перевірити, чи є в ній якийсь підрядок:

```
message = "hello world!"
hello = "hello"
print(hello in message)  # True - подстрока hello есть в строке "hello world!"

gold = "gold"
print(gold in message)  # False - подстроки "gold" нет в строке "hello world!"
```

Якщо нам треба навпаки перевірити, чи немає в наборі значень будь-якого значення, ми можемо використовувати модифікацію оператора - **not in**. Вона повертає **True**, якщо в наборі значень немає певного значення:

```
message = "hello world!"
hello = "hello"
print(hello not in message)  # False

gold = "gold"
print(gold not in message)  # True
```

3.Умовний оператор if. Імітація оператор вибору switch/case в мові Python.

if у Python - Оператор **if** є початком умовної конструкції. Далі йде умова, яка повертає логічне значення **True** (істина) або **False** (брехня). Завершується умова символом «двокрапка». Потім обов'язковий відступ у чотири пробіли, він показує, що рядки об'єднуються в один блок.

Оператор **if** призначено для виконання деякої послідовності дій у тому випадку, якщо істинною є зазначена умова.

Зверніть увагу!

Оператори записуються з обов’язковим однаковим відступом від лівого краю. Якщо оператор лише один, його можна записати після двокрапки в одному рядку з умовою..

Найпростіший приклад - це порівняння деякої змінної зі значеннями (спочатку розглянемо як це було б з **if/else**):

```
def load():
    print("Загружаем")
def save():
    print("Сохраняем")
def default():
    print("Неизвестно как обработать")

def main(value):
    if isinstance(value, str) and value == "load":
        load()
    elif isinstance(value, str) and value == "save":
        save()
    else:
        default()

main("load")
>>> Загружаем
main("save")
>>> Сохраняем
main("hello")
>>> Неизвестно как обработать
```

match/case:

```
def main(value):
    match value:
        case "load":
            load()
        case "save":
            save()
        case _:
            default()
main("load")
>>> Загружаем
main("save")
>>> Сохраняем
main(5645)
>>> Неизвестно как обработать
```

Стало помітно менше "and" і "==", вдалося позбутися зайвих перевірок на тип даних і код став більш зрозумілим, проте це лише найпростіший приклад, заглибимося далі. Припустимо, звідкись приходять дані у вигляді рядка, які записані з роздільником “~”, і заздалегідь відомо, що якщо даних було рівно 2 значення, то виконати одну дію, якщо 3 значення, то іншу дію:

```
def load(link):
    print("Загружаем", link)
    return "hello"
def save(link, filename):
    data = load(link)
    print("Сохраняем в", filename)
def default(values):
    print("Неизвестно как эти данные обработать")

def main(data_string):
    values = data_string.split("~")
    if isinstance(values, (list, tuple)) and len(values) == 2 and values[0] == "load":
        load(values[1])
    elif isinstance(values, (list, tuple)) and len(values) == 3 and values[0] == "save":
        save(values[1], values[2])
    else:
        default(values)

main("load~http://example.com/files/test.txt")
>>> Загружаем http://example.com/files/test.txt
main("save~http://example.com/files/test.txt~file.txt")
>>> Загружаем http://example.com/files/test.txt
>>> Сохраняем в file.txt
main("use~http://example.com/files/test.txt~file.txt")
>>> Неизвестно как эти данные обработать
main("save~http://example.com/files/test.txt~file.txt~file2.txt")
>>> Неизвестно как эти данные обработать
```

match/case:

```
def main(data_string):
    values = data_string.split("~")
    match values:
        case "load", link:
            load(link)
        case "save", link, filename:
            save(link, filename)
        case _:
            default(values)
main("load~http://example.com/files/test.txt")
>>> Загружаем http://example.com/files/test.txt
main("save~http://example.com/files/test.txt~file.txt")
>>> Загружаем http://example.com/files/test.txt
>>> Сохраняем в file.txt
main("use~http://example.com/files/test.txt~file.txt")
>>> Неизвестно как эти данные обработать
main("save~http://example.com/files/test.txt~file.txt~file2.txt")
>>> Неизвестно как эти данные обработать
```

Якщо є необхідно завантажити кілька файлів:

```
def load(links):
    print("Загружаем", links)
    return "hello"
def main(data_string):
    values = data_string.split("~")
    match values:
        case "load", *links:
            load(links)
        case _:
            default(values)
main("load~http://example.com/files/test.txt~http://example.com/files/test1.txt")
>>> Загружаем ['http://example.com/files/test.txt', 'http://example.com/files/test1.txt']
```

Match/case сам вирішує проблему з перевіркою типів даних, з перевіркою значень та їх кількістю, що дозволяє спростити логіку та збільшити читання коду. І дуже зручно, що можна оголошувати змінні та поміщати в них значення прямо у гілці **case** без використання моржового оператора.

Розглянемо приклад, коли необхідно використовувати оператор **"або"** у прикладі. Припустимо, надходить запит від користувача з правами, і необхідно перевірити, чи може цей користувач виконувати поточну дію:

```
def main(data_string):
    values = data_string.split("~")
    match values:
        case name, "1"|"2" as access, request:
            print(f"Пользователь {name} получил доступ к функции {request} с правами {acce
        case _:
            print("Неудача")
main("Daniil~2~load")
>>> Пользователь Daniil получил доступ к функции load с правами 2
main("Kris~0~save")
>>> Неудача
```

У такому разі символ "|" виступає у ролі логічного **“чи”,** а значення прав доступу змінну **access** було записано з допомогою оператора **"as".** Розберемо аналогічний приклад, але як аргумент будемо розглядати словник:

```
def main(data_dict):
   match data_dict:
       case {"name": str(name), "access": 1|2 as access, "request": request}:
           print(f"Пользователь {name} получил доступ к функции {request} с правами {acce
       case _:
           print("Неудача")
main({"name": "Daniil", "access": 1, "request": "save"})
>>> Пользователь Daniil получил доступ к функции save с правами 1
main({"name": ["Daniil"], "access": 1, "request": "save"})
>>> Неудача
main({"name": "Kris", "access": 0, "request": "load"})
>>> Неудача
```

Блок match/case для классов:

```
class UserRequest:
   def __init__(self, name, access, request):
       self.name = name
       self.access = access
       self.request = request
def main(data_class):
   match data_class:
       case UserRequest(name=str(name), access=1|2 as access, request=request):
           print(f"Пользователь {name} получил доступ к функции {request} с правами {acce
       case _:
           print("Неудача")
main(UserRequest("Daniil", 1, "delete"))
>>> Пользователь Daniil получил доступ к функции delete с правами 1
main(UserRequest(1234, 1, "delete"))
>>> Неудача
main(UserRequest("Kris", 0, "save"))
>>> Неудача
```

@ZTUEDUUA

ЖИТОМИРСЬКА ПОЛІТЕХНІКА

ДЕРЖАВНИЙ УНІВЕРСИТЕТ

- **Розвиваємо лідерів**
- **Створюємо інновації**
- **Змінюємо світ на краще**