Основные операторы языка (продолжение)

Условный оператор

Раздаточный материал № 22

```
#1
if n < 100:
        b = n + a #отступ является обязательным, т.к. формирует тело условного
        оператора
print(b) # оператор print не является телом условного оператора

#2
tovar1 = 50
tovar2 = 32
if tovar1 + tovar2 > 99 :
        print("99 рублей недостаточно")
else:
        print("Чек оплачен")

#3

a = 5 > 0 # подвыражение 5 > 0 выполнится первым, после чего его результат будет
            присвоен переменной a
if a:
        print(a)

if a > 0 and a < b:
        print(b - a)

if 0 < a < b:
        print(b - a)
```

Множественное ветвление предполагает выбор больше, чем из двух путей, например, из трех, четырех или даже пяти.

Раздаточный материал № 23

```
old = int(input('Ваш возраст: '))
print('Рекомендовано:', end=' ')
if 3 <= old < 6:
        print('"Заяц в лабиринте"')
elif 6 <= old < 12:
        print('"Марсианин"')
elif 12 <= old < 16:
        print('"Загадочный остров"')
elif 16 <= old:
        print('"Поток сознания"')
```

Тернарный оператор

по логике вычисления очень похож на условный оператор и имеет формат

Раздаточный материал № 24

```
a if условие else b
```

Сначала оценивается условие, затем вычисляется ровно одно из a или b и возвращается на основе логического значения условие. Если условие оценивается как True, то a оценивается и возвращается, но b игнорируется, или же, когда b оценивается и возвращается, но a игнорируется.

Раздаточный материал № 25

```
a = 50
b = 100
c = 40
max = a if a > b else b
max = c if c > max else max
print(max)
```

Тернарный оператор может быть использован как самостоятельно, так и в составе выражения, в функции, в другом операторе. В качестве операнда может выступать только выражение, но не оператор.

Раздаточный материал № 26

```
# Дано целое число. Если оно является положительным,
# то прибавить к нему 20, в противном случае вычесть из него 5/
# Результат не сохраняется

c = int(input('Введи число:     '))
print('Результат =  ', c + 20 if c >= 0 else c - 5)

# Дано целое число. Если оно является положительным,
# то прибавить к нему 20, в противном случае вычесть из него 5/
# Результат сохраняется

c = int(input('Введи число:     '))
f = c + 20 if c >= 0 else c - 5
print('Результат =  ', f)
```

Оператор while

многократно выполняет блок операторов (обычно с отступом) до тех пор, пока проверка в заголовочной части оценивается как истинное значение. Это называется “циклом”, потому что управление продолжает возвращаться к началу оператора, пока проверка не даст ложное значение. Когда результат проверки становится ложным, управление переходит на оператора, следующий после блока while. Если проверка оценивается в ложное значение с самого начала, тогда тело цикла никогда не выполнится, и оператор while пропускается.

Раздаточный материал № 27

```
while проверка:
        операторы
        if проверка: break          # Выход из цикла с пропуском else, если есть
        if проверка: continue       # Переход на проверку в начале цикла
else:
        операторы                   # Выполняется, если не было break
```

Блок else цикла выполняется тогда и только тогда, когда происходит нормальный выход из цикла (т.е. без выполнения оператора break).

Оператор pass — это заполнитель, обозначающий отсутствие действий, который используется в ситуациях, когда синтаксис требует оператора, но нет возможности

выполнить что-либо полезное. Например, бесконечный цикл, который на каждом проходе ничего не делает или иногда оператор pass обозначает место, подлежащее заполнению в будущем, что служит временной заглушкой для тел функций:

Раздаточный материал № 28

```
def fund () : pass      # Позже поместить сюда реальный код
def func2(): pass
```

Нельзя оставить тело функции пустым, не получив синтаксической ошибки, поэтому используется pass.

```
while True: pass      # Для прекращения работы нажмите <Ctrl+C>!
```

Аналог pass – троеточие

Раздаточный материал № 29

```
def func1(): ...
func1()            # При вызове ничего не делает
```

Оператор continue вызывает немедленный переход в начало цикла.

Раздаточный материал № 30

```
t = 10
while t:
    t -= 1
    if t % 2 != 0: continue # пропуск нечетных чисел
    print(t, end=' ')
```

Оператор break вызывает немедленный выход из цикла.

Раздаточный материал № 31

```
while True:
    name = input('Enter name: ')
    if name == 'stop': break # при вводе stop – выход из цикла
    age = input ('Enter age: ')
    print('Hello', name, '=>', int(age) ** 2)
```

Обработка исключений

Исключениями являются:

1. Несоответствие типов данных - TypeError.
2. Попытка деления на 0 - ZeroDivisionError.
3. Ошибка значения - ValueError
4. И др.

Обработку исключений рекомендуется производить чтобы исключить некорректные действия пользователя. Для этого используется оператор try – except.

Раздаточный материал № 32

```
try:
    n = int(input("Введите целое число: "))
    print("Удачно")
except:
    print("Что-то пошло не так")
```

Если в теле try исключения не возникает, то тело ветки except не выполняется. В теле try могут возникнуть любые исключения, поэтому необходимо указывать тип исключения после `except:`

Раздаточный материал № 33

```
try:
    n = int(input("Введите целое число: "))
    print("Удачно")
except ValueError:
    print("Что-то пошло не так")
```

На каждое исключение пишется собственная ветка except.

У оператора обработки исключений, кроме except, могут быть еще ветки finally и else (не обязательно обе сразу). Тело finally выполняется всегда, независимо от того, выполнялись ли блоки except в ответ на возникшие исключения или нет. Тело else сработает, если исключений в try не было, т. е. не было переходов на блоки except.

Также исключение может возникнуть в блоках except, else или finally, и тогда им нужен собственный обработчик. В этом случае except начинается с отступа.

Исключения и while

Раздаточный материал № 34

```
n = input("Введите целое число: ")
while type(n) != int:
    try:
        n = int(n)
    except ValueError:
      print("Неправильно ввели!")
      n = input("Введите целое число: ")

if  not(math.fmod(n, 2)) :
        print("Четное")
else:
        print("Нечетное")
```

Фронтальный опрос:

- Описать работу условного оператора.
- Описать работу тернарного оператора.
- Описать работу оператора while.
- Назначение оператора pass.
- Назначение оператора *continue*
- Назначение оператора *break*
- Назначение оператора try – except.

Подготовка к тестированию по PEP8 – стиль кода в Python.