PYTHON

Тема2: Функции.

Передача аргументов

- Аргументы передаются через автоматическое присваивание объектов локальным переменным.
- Операция присваивания именам аргументов внутри функции не оказывает влияния на вызывающую программу.
- Изменение внутри функции аргумента, который является изменяемым объектом, может оказывать влияние на вызывающую программу.

- ✓ **Неизменяемые объекты передаются «по значению»**.
- ✓ **Изменяемые объекты передаются «по указателю»**.

PYTHON

Тема2: Функции.

Передача аргументов

Действия, которые выполняет интерпретатор при сопоставлении аргументов перед присваиванием:

1. Сопоставление не именованных аргументов по позициям.
2. Сопоставление именованных аргументов по именам.
3. Сопоставление дополнительных не именованных аргументов с кортежем *name.
4. Сопоставление дополнительных именованных аргументов со словарем **name.
5. Сопоставление значений по умолчанию с отсутствующими именованными аргументами.

PYTHON

Тема2: Функции.

Передача аргументов

```
# Комбинирование именованных аргументов и значений по умолчанию:
def func(spam, eggs, toast=0, ham=0):    # Первые 2 являются
    print(spam, eggs, toast, ham)        # обязательными

func(1, 2)                               # вывод(1,2,0,0)
func(1, ham=1, eggs=0)                   # вывод(1,0,0,1)
func(spam=1, eggs=0)                     # вывод(1,0,0,0)
func(toast=1, eggs=2, spam=3)            # вывод(3,2,1,0)

func(1, 2, 3, 4)                         # вывод(1,2,3,4)

#Комбинирование:
def f(a, *pargs, **kargs):
    print(a, pargs, kargs)

f(1, 2, 3, x=1, y=2)
#1 (2, 3) {'y': 2, 'x': 1}
```

Передача аргументов

```
def func(a, b, c, d): print(a, b, c, d)

args = (1, 2)
args += (3, 4)
func(*args)

# 1 2 3 4



#**:
args = {'a': 1, 'b': 2, 'c': 3}
args['d'] = 4
func(**args)
# 1 2 3 4
```

можно очень гибко комбинировать в одном вызове обычные позиционные и именованные аргументы:

```
>>> func(*(1, 2), **{‘d’: 4, ‘c’: 4})
1 2 4 4
>>> func(1, *(2, 3), **{‘d’: 4})
1 2 3 4
>>> func(1, c=3, *(2,), **{‘d’: 4})
1 2 3 4
>>> func(1, *(2, 3), d=4)
1 2 3 4
>>> f(1, *(2,), c=3, **{‘d’:4})
1 2 3 4
```

PYTHON

Тема2: Функции.

Обобщенные способы вызова функций:

В программе можно использовать условную инструкцию **if** для выбора из множества функций и списков аргументов и вызывать любую из них единообразным способом...

```
if <test>:
   action, args = func1, (1,)        # Вызвать func1 с 1 аргументом
else:
   action, args = func2, (1, 2, 3)   # Вызвать func2 с 3 аргументами
...

action(*args) # Фактический вызов универсальным способом
```

PYTHON

Тема2: Функции.

Обобщенные способы вызова функций:

Пример, в следующем фрагменте реализована поддержка вызова произвольных функций с любым количеством любых аргументов, передавая все аргументы, которые были получены:

```
tracer(func, *pargs, **kargs):
    print('calling:', func.__name__)# Принимает произвольные
                                    # аргументы
    return func(*pargs, **kargs)# Передает все полученные
                                # аргументы
def func(a, b, c, d):
  return a + b + c + d

print(tracer(func, 1, 2, c=3, d=4))
```

PYTHON

Тема2: Функции.Возвращение результата.

Функция может возвращать результат. Для этого в функции используется оператор **return**, после которого указывается возвращаемое значение:

```
def exchange(usd_rate, money):
    result = round(money/usd_rate, 2)
    return result

result1 = exchange(60, 30000)
print(result1)
result2 = exchange(56, 30000)
print(result2)
result3 = exchange(65, 30000)
print(result3)
```

PYTHON

Тема2: Функции.Возвращение результата.

В Python функция может возвращать сразу несколько значений:

```
def create_default_user():
    name = "Tom"
    age = 33
    return name, age


user_name, user_age = create_default_user()
print("Name:", user_name, "\t Age:", user_age)
```

PYTHON

Тема2: Функции.Возвращение результата.

Функция может быть любой сложности и возвращать любые объекты (списки, кортежи, и даже функции!):

```
def newfunc(n):
    def myfunc(x):
        return x + n
    return myfunc

new = newfunc(100) # new - это функция
new(200)
```

300

PYTHON

Тема2: Функции.

Косвенный вызов функций:

```
def echo(message): # Имени echo присваивается объект функции
...     print(message)
...
echo('Direct call')
#Direct call
x = echo
x('Indirect call!')
#Indirect call
```

PYTHON

Тема2: Функции.

Функции легко можно передавать другим функциям в виде аргументов:

```
def indirect(func, arg):
...     func(arg) # Вызов объекта добавлением ()
...
indirect(echo, ‘Argument call!’) # Передача функции в функцию

Argument call!
```

Функция может и не заканчиваться инструкцией return, при этом функция вернет значение None:

```
def func():
...     pass
...
print(func())
None
```

PYTHON

Тема2: Функции.

Рекурсия

```
def fact(num):
    if num == 0:
        return 1 # По договоренности факториал нуля равен единице
    else:
        return num * fact(num - 1) # возвращаем результат произведения num и результата возвращенного функцией fact(num - 1)
```

PYTHON

Тема2: Функции.

Функция main

```
def main():
    say_hello("Tom")
    usd_rate = 56
    money = 30000
    result = exchange(usd_rate, money)
    print("К выдаче", result, "долларов")

 def say_hello(name):
    print("Hello,", name)

def exchange(usd_rate, money):
    result = round(money/usd_rate, 2)
    return result

# Вызов функции main
main()
```

PYTHON

Тема2: Функции.Анонимные функции, инструкция lambda.

Lambda-функция – это безымянная функция с произвольным числом аргументов и вычисляющая одно выражение. Тело такой функции не может содержать более одной инструкции (или выражения). Данную функцию можно использовать в рамках каких-либо конвейерных вычислений (например внутри **filter()**, **map()** и **reduce()**) либо самостоятельно, в тех местах, где требуется произвести какие вычисление, которые удобно “завернуть” в функцию.

```
func = lambda x, y: x + y
func(1, 2)
3
func('a', 'b')
'ab'
(lambda x, y: x + y)(1, 2)
3
(lambda x, y: x + y)('a', 'b')
'ab'
```

lambda функции, в отличие от обычной, не требуется инструкция return, а в остальном, ведет себя точно так же:

```
func = lambda *args: args
func(1, 2, 3, 4)
(1, 2, 3, 4)
```