

Python's basic constructs

September 16, 2010

1

Values

Numbers, letters, words, ... are collectively called *values*.

Example. The following are all values.

1823

3.141592653

'Trinity College'

September 16, 2010

2

Types

Values are categorized into different *types*: integers, floating-point (real) numbers, strings, ...

Each value belongs to a single type (i.e., no value belongs to more than one type).

September 16, 2010

3

Variables

A *variable* is a name that refers to a value.

An *assignment* statement creates a variable and assigns a value to it.

Example.

```
>>> n = 1823
>>> print n
1823
```

September 16, 2010

4

Variables and types

The *type* of a variable is the type of its value.

Example.

```
>>> n = 1823
>>> type(n)
<type 'int'>
```

September 16, 2010

5

Variable names

A variable can be named by any word (except for certain reserved words).

A variable name can contain any letter or digit, but it must begin with a letter.

To name a variable using more than one word, it is customary to use `_` to connect words.

September 16, 2010

6

Variable names

Example.

```
temp = 33
exam3 = 89
my_college = 'Trinity College'
```

September 16, 2010

7

Statements

A *statement* is the smallest unit of code that can be executed as a stand-alone instruction.

Example.

```
n = 1823
```

September 16, 2010

8

Expressions

An *expression* is a code fragment that has a value.

Example.

```
1823 + n (assuming n has a value)
```

September 16, 2010

9

Expressions

An *expression* is a code fragment that has a value.

Non example.

`n = 1823`

September 16, 2010

10

Statement and expressions

An *expression* is usually part of a statement.

Example. This statement

`n = 11 * (2 + 3)`

contains an expression `2 + 3`.

September 16, 2010

11

Functions

September 16, 2010

12

Functions

A **function** is a subprogram—a small program inside a program—with a name.

- Can be used anywhere anytime by calling its name.
- Most useful when we want to repeat the same basic task (e.g., rounding a real number to an integer).

September 16, 2010

13

Why functions?

- Make easier to read and debug.
- Make programs shorter by eliminating repetitive code; if you make a change, you only have to make it in one place.
- Can be used for other purposes in the future...

September 16, 2010

14

Functions

Example. Suppose you want to write a program to print out these famous lyrics:

```
Happy birthday to you!  
Happy birthday to you!  
Happy birthday, dear <someone>.  
Happy birthday to you!
```

September 16, 2010

15

Functions

If you know who <someone> is, say, Joe, you would just write:

```
print 'Happy birthday to you!'
print 'Happy birthday to you!'
print 'Happy birthday, dear Joe.'
print 'Happy birthday to you!'
```

But this code has a lot of repetition!

September 16, 2010

16

Functions

To save this repetition, we can define a function for printing "Happy birthday to you!" in the following way.

```
def happy():
    print 'Happy birthday to you!'
```

This function is named happy.

September 16, 2010

17

Functions

To *call* or *invoke* this function happy, all you need to do is:

```
>>> happy()
Happy birthday to you!
```

September 16, 2010

18

Functions

The lyrics for Joe can be simplified as:

```
happy()  
happy()  
print 'Happy birthday, dear Joe.'  
happy()
```

which will print:

September 16, 2010

19

Functions

```
Happy birthday to you!  
Happy birthday to you!  
Happy birthday, dear Joe.  
Happy birthday to you!
```

September 16, 2010

20

Functions

We can also make this a function as well:

```
def singJoe():  
    happy()  
    happy()  
    print 'Happy birthday, dear Joe.'  
    happy()
```

September 16, 2010

21

Functions

Invoking the function `singJoe` gives:

```
>> singJoe()  
Happy birthday to you!  
Happy birthday to you!  
Happy birthday, dear Joe.  
Happy birthday to you!
```

September 16, 2010

22

Functions

We can also make this a function for Ann:

```
def singAnn():  
    happy()  
    happy()  
    print 'Happy birthday, dear Ann.'  
    happy()
```

September 16, 2010

23

Functions

What does this do?

```
singJoe()  
print  
singAnn()
```

September 16, 2010

24

Functions

```
Happy birthday to you!  
Happy birthday to you!  
Happy birthday, dear Joe.  
Happy birthday to you!  
  
Happy birthday to you!  
Happy birthday to you!  
Happy birthday, dear Ann.  
Happy birthday to you!
```

September 16, 2010

25

Functions

I still see some repetition here. Can we make a generic function to sing happy birthday to anyone and use that for Joe and Ann somehow?

Yes, using a *parameter*—a variable whose value is specified by the caller (or user).

September 16, 2010

26

Functions

Consider this function for anyone:

```
def sing(anyone):  
    happy()  
    happy()  
    print 'Happy birthday, dear', anyone + '.'  
    happy()
```

Here, the value of anyone is to be specified by the caller of this function.

September 16, 2010

27

Functions

Invoking the function `sing` with `'Joe'` gives:

```
>> sing('Joe')
Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Joe.
Happy birthday to you!
```

Here, `'Joe'` is passed as an argument to `sing`'s parameter `anyone`.

September 16, 2010

28

Functions

Invoking the function `sing` with `'Ann'` gives:

```
>> sing('Ann')
Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Ann.
Happy birthday to you!
```

Here, `'Ann'` is passed as an argument to `sing`'s parameter `anyone`.

September 16, 2010

29

Functions

Now, what does this do?

```
sing('Joe')
print
sing('Ann')
```

September 16, 2010

30

Functions

```
Happy birthday to you!  
Happy birthday to you!  
Happy birthday, dear Joe.  
Happy birthday to you!  
  
Happy birthday to you!  
Happy birthday to you!  
Happy birthday, dear Ann.  
Happy birthday to you!
```

September 16, 2010

31

Python's built-in functions

Programmers often write their own functions, but there are also functions within Python—*built-in functions*—that have already been written for us to use.

September 16, 2010

32

The function type

The `type` command is actually a built-in function that returns the type of a given value.

Example.

```
>>> n = 1823  
>>> type(n)  
<type 'int'>
```

September 16, 2010

33

Type conversion functions

Python has functions to convert values of one type to those of another.

Example.

```
>>> int('1823')
1823
```

September 16, 2010

34

Type conversion functions

Python has functions to convert values of one type to those of another.

Example.

```
>>> int(3.141592653)
3
```

September 16, 2010

35

Type conversion functions

Python has functions to convert values of one type to those of another.

Example.

```
>>> float(1823)
1823.0
```

September 16, 2010

36

Type conversion functions

Python has functions to convert values of one type to those of another.

Example.

```
>>> float('3.141592653')
3.141592653
```

September 16, 2010

37

Type conversion functions

Python has functions to convert values of one type to those of another.

Example.

```
>>> str(3.141592653)
'3.141592653'
```

September 16, 2010

38

Math functions

Python has a module of common mathematical functions, named `math`.

Example.

```
>>> import math
>>> math.sqrt(2)
1.4142135623730951
```

September 16, 2010

39
