Data Types, Variables and Expressions

Prof. Harald Gall,

University of Zurich, Institute of Informatics

Outline

- data types
- variables
- operators
- expressions
- strings
- syntax and semantics
- comments
- example exercises

Data types

- Programs mainly manipulate **data** (**values or objects**).
- Each values has a specific type which defines how to interpret the value and what operations (**operators**) can be performed on it

```
>>> print(4)
4
>>> print('Hello, World!')
Hello World
```

Scalar Value Types

- *hold a single value*
- Numeric types: **int** (3, 5, etc.), **float** (3.5, 123.23), **complex** (2 + 3j)
- NoneType: **None** (represents the absence of a value)
- Boolean types: **True**, **False**

Compound Value Types

- *hold multiple independent values*
- **Strings**: 'Hello, World!', "Hello, World!"
- **Lists**, **dictionaries**, **tuples** (more on that later)
- **User defined types** (more on that later) **Classes & Structs**

Numeric types - Examples

```
>>> print(4)
4
>>> print(4.5)
4.5
>>> print(2 + 3j)
(2 + 3j)
```

Numeric types – Checking the type

```
>>> type(4)
<class 'int'>
>>> type(4.5)
<class 'float'>
>>> type(2 + 3j)
<class 'complex'>
```

Numeric types - Operators

Operation	Result
x + y	sum of x and y
x - y	difference of x and y
x * y	product of x and y
x / y	float division of x and y
x // y	integer division of x and y
x % y	remainder of x / y
-x	x negated
+x	x unchanged
int(x)	x converted to integer
long(x)	x converted to long integer
float(x)	x converted to floating point
complex(re, im)	a complex number with real part re, imaginary part im(defaults to zero)
x ** y or pow(x, y)	x to the power of y

Numeric Operators - Examples

```
>>> 3 + 5
8
>>> 3 / 2
1.5
>>> 3 // 2
1
>>> 2 ** 3
8
```

Boolean type – Operators

Operation	Result
x and y	if x is false, then y, else x
x or y	if x is false, then x, else y
not x	if x is false, then True, else False

Boolean operators – Examples

```
>>> True and False
False
>>> True or False
True
>>> not True
False
```

Variables

- A variable binds a name to a specific value through the assignment operation

```
>>> message = 'Hello, World!'

>>> n = 100

>>> b = True
```

Variables

- the variable name is then used to refer to the stored value

```
>>> message = 'Hello, World!'

>>> print(message)

Hello, World!
```

Variables

- a variable name can be re-assigned

```
>>> message = 'Hello, World!'
>>> print(message)
Hello, World!
>>> message = "Hello, Bob!"
>>> print(message)
Hello, Bob!
```

Variable names

- Can contain characters, numbers and '_'
- Case sensitive: **Banana** is not the same as **banana**
- Choose meaningful names
- Naming convention: lowercase with words separated by underscores
- Cannot use reserved keywords as variable names

Reserved keywords

```
and       assert    break     class     continue  def
del       elif      else      except    exec
finally   for       from      global    if        import
in        is        lambda    not       or        pass
print     raise     return    try       while     yield
```

Expressions

- We can combine values, variables and operators to build complex expressions that are evaluated by the interpreter and produce a result

```
>>> x = 2
>>> y = 3
>>> z = 4
>>> x + y * z + 100
114
```

Präzendenz

How are expressions evaluated?

The **operator precedence** determines the order in which operators are evaluated. Operators are evaluated in the following order:

- Parentheses – ()
- Exponentiation - **
- Multiplication and Division - *, /, //
- Addition and Subtraction - +, -

- Operators with the same precedence are evaluated from left to right

Operators precedence - Example

```
(2 * 3 - 1) ** 2 + 1
```

Operators precedence - Example

```
(2 * 3 - 1) ** 2 + 1
(2 * 3 - 1) ** 2 + 1
```

Operators precedence - Example

```
(2 * 3 - 1) ** 2 + 1
(2 * 3 - 1) ** 2 + 1
(2 * 3 - 1) ** 2 + 1
```

Operators precedence - Example

```
(2 * 3 - 1) ** 2 + 1
(2 * 3 - 1) ** 2 + 1
(2 * 3 - 1) ** 2 + 1
(6 - 1) ** 2 + 1
```

Operators precedence - Example

```
(2 * 3 - 1) ** 2 + 1
(2 * 3 - 1) ** 2 + 1
(2 * 3 - 1) ** 2 + 1
(6 - 1) ** 2 + 1
```

Operators precedence - Example

```
(2 * 3 - 1) ** 2 + 1
(2 * 3 - 1) ** 2 + 1
(2 * 3 - 1) ** 2 + 1
(6 - 1) ** 2 + 1
5 ** 2 + 1
```

Operators precedence - Example

```
(2 * 3 - 1) ** 2 + 1
(2 * 3 - 1) ** 2 + 1
(2 * 3 - 1) ** 2 + 1
(6 - 1) ** 2 + 1
5 ** 2 + 1
```

Operators precedence - Example

```
(2 * 3 - 1) ** 2 + 1
(2 * 3 - 1) ** 2 + 1
(2 * 3 - 1) ** 2 + 1
(6 - 1) ** 2 + 1
5 ** 2 + 1
25 + 1
```

Operators precedence - Example

```
(2 * 3 - 1) ** 2 + 1
(2 * 3 - 1) ** 2 + 1
(2 * 3 - 1) ** 2 + 1
(6 - 1) ** 2 + 1
5 ** 2 + 1
25 + 1
```

Operators precedence - Example

```
(2 * 3 - 1) ** 2 + 1
(2 * 3 - 1) ** 2 + 1
(2 * 3 - 1) ** 2 + 1
(6 - 1) ** 2 + 1
5 ** 2 + 1
26
```

Strings

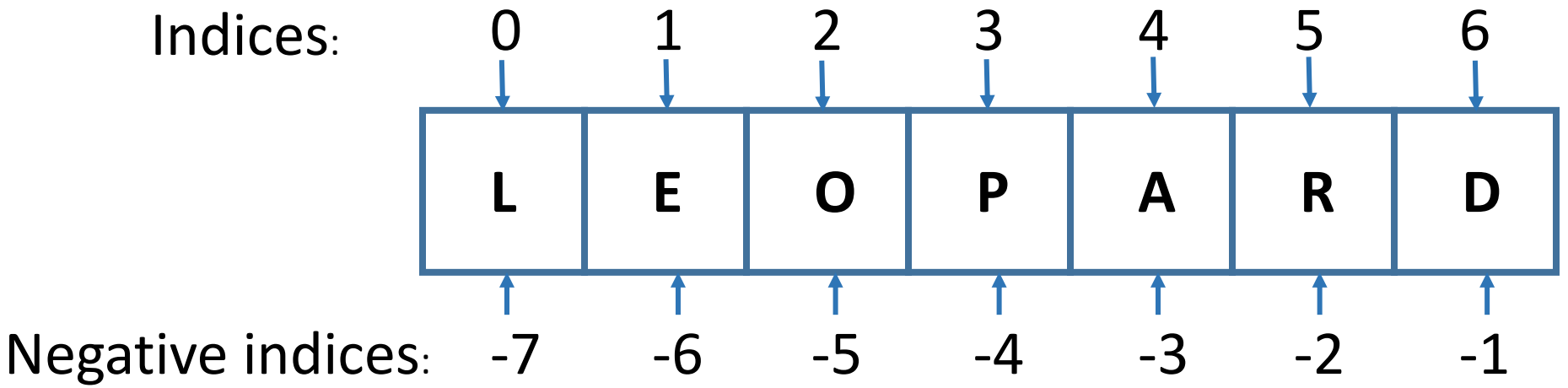
- a compound data type (made up of a sequence of characters)

```
>>> animal = 'dog'
>>> print(animal)
dog
>>> animal = "cat"
>>> print(animal)
cat
```

Strings – Accessing the characters

```
>>> animal = 'leopard'
>>> animal[0]
l
>>> animal[1]
e
>>> animal[-1]
d
>>> animal[10]
IndexError: string index out of range
```

Strings – Accessing the characters



String slicing

```
>>> animal = 'leopard'
>>> animal[0:3]
'leo'
>>> animal[:3]
'leo'
>>> animal[5:7]
'rd'
>>> animal[5:]
'rd'
```

String operators

Operation	Result
x + y	concatenation
x * n	x is a string, n is a number, repeats string x n times

String operators - Examples

```
>>> firstname = 'Alice'
>>> lastname = 'Smith'
>>> firstname + lastname
'AliceSmith'
>>> firstname + ' ' + lastname
'Alice Smith'
>>> firstname * 3
'AliceAliceAlice'
>>> 3 * lastname
'SmithSmithSmith'
```

String formatting

- you can build complex string expression using the concatenation operator (+), but this can become cumbersome
- a better way is to use the formatting operator (%) or the format method
- check link for full details:

  https://docs.python.org/2.4/lib/typesseq-strings.html

String formatting operator - Examples

```
>>> name = 'Alice'
>>> 'Hello, %s!' % name
'Hello, Alice!'
>>> age = 25
>>> 'Hello, %s! You are %d years old.' % (name, age)
'Hello, Alice! You are 25 years old.'
```

Syntax

- the **syntax** of a programming language is the set of rules that define what combination of symbols (values, variables, expression, operators, etc.) are considered valid
- if you do not follow the rules the interpreter will signal a **syntax error**

```
>>> 'Hello' + 3
TypeError: Can't convert 'int' object to str implicitly
>>> print(name)
NameError: name 'name' is not defined
```

Semantics

- **semantics** is the meaning associated with a syntactically correct string of symbols without syntactical errors

Types of errors

- syntactical errors (will be signaled by the interpreter)
- semantical error (some errors might be signaled)
- different meaning than what the programmer actually intended (the program crashes, runs forever or returns a different result than what was actually intended)

Getting data from the user

- to get data from the user you can use the input method
- this will return a string containing what the user typed in the terminal

```
>>> name = input('Your name:')
Your name: Bob
>>> print('Hello, %s!' % name)
'Hello, Bob!'
```

Comments

- comments are explanations or annotations of source code written by the programmer
- they are generally ignored by the interpreter (compiler) and should make the code easier to understand

```
# ask for the user name
username = input('Your name: ')
msg = 'Hello, ' + username + '!' # Build message
print(msg)
```

Exercise 1

Using Python as a calculator - compute the following expressions with Python:

- $2 + 2 + 5 \times 10^2$
- $(1 + 10^3) \times (2 + 5^2)$
- $\sqrt[2]{(1 + 10)} - (1 + 10)^3$

Exercise 1 - $2 + 2 + 5 \times 10^2$

```
>>> 2 + 2 + 5 * 10 ** 2
504
```

Exercise 1 - $(1 + 10^3) \times (2 + 5^2)$

```
>>> (1 + 10 ** 3) * (2 + 5 ** 2)
27027
```

Exercise 1 - $\sqrt[2]{(1+10)} - (1+10)^3$

```
>>> (1 + 10) ** 1/2 - (1 + 10) ** 3
-1325.5
```

Exercise 2

Ask from the user the following information: name, age, occupation and print the following message using the provided values:

Hi NAME! I see you are AGE years old and work as a OCCUPATION.

Ask for the name

```
input('What is your name? ')
```

Store what the user has typed in a variable

```
name = input('What is your name? ')
```

Ask for the age and occupation

```
name = input('What is your name? ')
age = input('How old are you? ')
occupation = input('What is your occupation? ')
```

Build the message – option 1

```
name = input('What is your name? ')
age = input('How old are you? ')
occupation = input('What is your occupation? ')
msg = 'Hi ' + name + '! I see you are ' + age + '
old and work as a ' + occupation + '.'
print(msg)
```

Build the message – option 2

```
name = input('What is your name?')
age = input('How old are you?')
occupation = input('What is your occupation?')
print('Hi %s! I see you are %s years old and work as
a %s.' % (name, age, occupation))
```

Exercise 3

Solve a quadratic equation: $a * x^2 + b * x + c = 0$

Ask from the user the values for a, b and c and print out the result for x.

Ask for the input values

```
# input returns a string, so we have to convert
# all values to float
a = float(input('a = '))
b = float(input('b = '))
c = float(input('c = '))
```

Solve using the quadratic formula

```
a = float(input('a = '))
b = float(input('b = '))
c = float(input('c = '))


x_1 = (-b + (b ** 2 - 4 * a * c) ** (1/2.0)) / (2*a)
x_2 = (-b - (b ** 2 - 4 * a * c) ** (1/2.0)) / (2*a)
```

Print the results

```
a = float(input('a = '))
b = float(input('b = '))
c = float(input('c = '))


x_1 = (-b + (b ** 2 - 4 * a * c) ** (1/2.0)) / (2*a)
x_2 = (-b - (b ** 2 - 4 * a * c) ** (1/2.0)) / (2*a)
print('Solutions: %.2f, %.2f'% (x_1, x_2))
```

What happens if you try to compute the square root of a negative number?

```
>> a = 4
>> b = 2
>> c = 1
>> x_1 = (-b + (b ** 2 - 4 * a * c) ** (1/2.0)) / (2 * a)
Traceback (most recent call last):
      File "<stdin>", line 1, in <module>
ValueError: negative number cannot be raised to a fractional
power
```

Exercise 4

Display the current time in Greenwich Mean Time (GMT) in the format hh:mm:ss using the `time.time()` function

(https://docs.python.org/3/library/time.html#time.time).

Note:

- before being able to use `time.time()`, you have to import the module time with the following instruction:

  ```
  import time
  ```

- `time.time()` returns the time in seconds since the epoch as a floating point number in Greenwich Mean Time.

Modules – Short Explanation

- modules contain already implemented functionality and can be used by simply typing `import module_name`
- they typically contain functions (next lecture) and variable definitions
- for now think of functions as a black-box, by calling a function (writing `time.time()`) you can obtain a result that you can use for your purposes

```
current_seconds = time.time()
```

First import the time module

```
import time
```

Save the current time in a variable

```
import time


# total number of seconds since 01.01.1970
total_sec = time.time()
```

Print it to see what it contains

```
import time


total_sec = time.time()
print(total_sec)
```

Convert to int

```
import time


total_sec = time.time()
total_sec = int(total_sec)
print(total_sec)
```

Compute the current second

```
import time


total_sec = time.time()
current_sec = total_sec % 60
print(current_sec)
```

Modulo

Compute total minutes

```
import time


total_sec = time.time()
current_sec = total_sec % 60
total_min = total_sec // 60
print(total_min)
```

Compute current minute

```
import time


total_sec = time.time()
current_sec = total_sec % 60
total_min = total_sec // 60
current_min = total_min % 60
print(current_min)
```

Compute total hours

```
import time


total_sec = time.time()
current_sec = total_sec % 60
total_min = total_sec // 60
current_min = total_min % 60
total_hours = total_min // 60
print(total_hours)
```

Compute current hour

```
import time

total_sec = time.time()
current_sec = total_sec % 60
total_min = total_sec // 60
current_min = total_min % 60
total_hours = total_min // 60
current_hour = total_hours % 24

print(current_hour)
```

Print current time as hh:mm:ss

```
import time

total_sec = time.time()
current_sec = total_sec % 60
total_min = total_sec // 60
current_min = total_min % 60
total_hours = total_min // 60
current_hour = total_hours % 24

print('%02d:%02d:%02d' % (current_hour, current_min,
current_sec))
```

Exercise 5

Vending machine change: ask the user for an amount between 1 cent and 99 cents. Print out a combination of coins equal to that amount (quarters – 25 cents, dime – 10 cents, nickel – 5 cents and pennies – 1 cent)

Ask for the amount

```
amount = int(input('amount: '))
```

Check to see if the right amount was saved

```
amount = int(input('amount: '))
print(amount)
```

Compute quarters

```
amount = int(input('amount: '))
quarters = amount // 25
print(quarters)
```

Compute the remaining cents

```
amount = int(input('amount: '))
quarters = amount // 25
rest_amount = amount % 25
print(rest_amount)
```

Compute dimes

```
amount = int(input('amount: '))
quarters = amount // 25
rest_amount = amount % 25
dimes = rest_amount // 10
print(dimes)
```

Compute the remaining cents

```
amount = int(input('amount: '))
quarters = amount // 25
rest_amount = amount % 25
dimes = rest_amount // 10
rest_amount %= 10
print(rest_amount)
```

Compute nickels

```
amount = int(input('amount: '))
quarters = amount // 25
rest_amount = amount % 25
dimes = rest_amount // 10
rest_amount %= 10
nickels = rest_amount // 5
print(nickels)
```

Compute remaining

```
amount = int(input('amount: '))
quarters = amount // 25
rest_amount = amount % 25
dimes = rest_amount // 10
rest_amount %= 10
nickels = rest_amount // 5
rest_amount %= 5
print(rest_amount)
```

Compute pennies

```
amount = int(input('amount: '))
quarters = amount // 25
rest_amount = amount % 25
dimes = rest_amount // 10
rest_amount %= 10
nickels = rest_amount // 5
rest_amount %= 5
pennies = rest_amount
```

Print change

```
amount = int(input('amount: '))
quarters = amount // 25
rest_amount = amount % 25
dimes = rest_amount // 10
rest_amount %= 10
nickels = rest_amount // 5
rest_amount %= 5
pennies = rest_amount
print('%d quarters %d dimes %d nickels %d pennies' %
(quarters, dimes, nickels, pennies))
```