VALUES, EXPRESSIONS, VARIABLES, LOGIC

Dr Kirill Sidorov
sidorovk@cardiff.ac.uk

School of Computer Science and Informatics
Cardiff University

Two ways to use an interpreter:

- Read-evaluate-print loop (REPL) = interactive mode
  - The interpreter executes Python commands as you enter them and immediately outputs the results (if any).
- File mode = non-interactive mode = batch mode
  - The interpreter executes an entire program stored in a file (or files).
  - By convention, we add the extension `.py` to the names of files that store Python programs.
  - A program can be stored in a file(s) and executed whenever you want to.
- REPL is useful for playing with the interpreter, but for long programs we use the non-interactive mode (file mode).

Example

Using Python as an interactive calculator.

```
>>> 2+3
5
>>> 4*7
28
>>> 2 ** 5
32
>>> 4+3*2
10
```

- All values (bits of data that a program manipulates) belong to a certain data type:
  - `5` → integer (`int`)
  - `"Hello, World!"` → string (`str`)
  - `4.33` → floating point number (`float`)
- A data type determines:
  - How the values of this type are stored in memory.
  - What are the possible values.
  - What operations can preformed on values of this type.

In Python, you can find the type of any value using the command `type()`.

For example, these are some of the types that Python knows:

In Python, you can find the type of any value using the command `type()`.

For example, these are some of the types that Python knows:

```
type(5) → <type 'int'>
```

In Python, you can find the type of any value using the command `type()`.

For example, these are some of the types that Python knows:

```
type(5) → <type 'int'>
type("Hello, World!") → <type 'str'>
```

In Python, you can find the type of any value using the command `type()`.

For example, these are some of the types that Python knows:

```
type(5) → <type 'int'>
type("Hello, World!") → <type 'str'>
type(4.33) → <type 'float'>
```

In Python, you can find the type of any value using the command `type()`.

For example, these are some of the types that Python knows:

```
type(5) → <type 'int'>
type("Hello, World!") → <type 'str'>
type(4.33) → <type 'float'>
type(True) → <type 'bool'>
```

- Operators are symbols which denote operations to be carried out on values.

- A few examples of operators:

**+ − / * % < >= **

- The operator %, for instance, is used to find the remainder after division:

```
17 % 5 = 2
```

- The operator ** is used to compute powers (exponentiate):

```
3 ** 5 = 243
```

- When more than one operator is used in an expression, the order in which the prescribed operations will be carried out is important!

Example

$$3 * 2 + 7 \% 3 = ?$$

- When more than one operator is used in an expression, the order in which the prescribed operations will be carried out is important!

Example

$$3 * 2 + 7 \% 3 = ?$$

- Every language, including Python, has a set of rules that describe in what order are the operators evaluated. This is called operator precedence or order of operations.

- Below (some of) the operators are shown in descending order of precedence:

  ```
  ( )
  **
  *  /  %
  +  -
  ```

- Operators higher up in this list will be executed first.
- Operators on the same level in this list will be executed from left to right.

- Below (some of) the operators are shown in descending order of precedence:

  ```
  ( )
  **
  *  /  %
  +  -
  ```

- Operators higher up in this list will be executed first.
- Operators on the same level in this list will be executed from left to right.

Example

$$3 * 2 + 7 \% 3 = 7$$

```
( )
**
* / %
+ -
```

The rules can be remembered with PEMDAS mnemonic:

- Parentheses have the highest precedence. What is inside the parentheses is evaluated first: `4 * (4 - 2) = 8`.
- Exponents (powers), then
- Multiply, Divide, (and Remainder), then
- Add or Subtract.
- Otherwise just from left to right.

- Logical (Boolean) values are used to represent the logical concepts of a statement being “true” of “false”.
- In Python, Boolean values are denoted as: `True` and `False`. (Note the capital first letter.)
- Python supports the following logical operations: `and`, `or`, `not`.

Truth tables for the logical operations

A	not A
False	True
True	False

A	B	A and B
False	False	False
False	True	False
True	False	False
True	True	True

A	B	A or B
False	False	False
False	True	True
True	False	True
True	True	True

```
>>> not True
False
>>> False and True
False
>>> (not False) and (False or True)
True
>>> (False or True) and (False or (True and True))
True
>>> ((not False) and (not True)) or
    ((not True) and (not False))
False
```

- Certain operations in Python evaluate to Boolean values.
- A good example is comparison operations: `<`, `>`, `==` (equals), `!=` (not equals), `<=`, `>=`.
- Precedence (higher to lower): arithmetic, comparisons, `not`, `and`, `or`.

Example

```
>>> 4 > 5
False
>>> (12 % 5) < 5
True
>>> 3 + 4 == 4 + 3
True
>>> ((1 > 2) or (3 < 4)) and (5 <= 5)
True
>>> (2 < 5) == (3 < 4)
True
```

Often, you need to convert from one data type to another. The following functions accomplish this.

Example

```
>>> str(12)
'12'
>>> str(3.14)
'3.14'
>>> int('42')
42
>>> float('2.71')
2.71
```

What about converting to and from booleans?

- A variable is a human-friendly name that refers to a value.
- An assignment statement creates new variables and assigns the values (initialises them). In other words, assignment binds a name to a value:

- A variable is a human-friendly name that refers to a value.
- An assignment statement creates new variables and assigns the values (initialises them). In other words, assignment binds a name to a value:

Example

```
year = 2020
temperature = 36.6
covid = True
```

- Try to choose meaningful names.
  - Variables exist for the benefit of the human!
- Ideally, a variable name should be describing what the variable is used for.
- In Python, variable names may contain letters, digits, and underscore.
- But, they must begin with a letter.
- Variable names are case-sensitive. So, `pitch` is not the same as `Pitch`.

- Variable names can be arbitrarily long.
- They can contain multiple words.
- To use multiple words, separate each word with the underscore character: _

- Variable names can be arbitrarily long.
- They can contain multiple words.
- To use multiple words, separate each word with the underscore character: _

Example

```
post_code = "CF24 3AA"
total_mark = 75
speed_of_light = 299792458
```

Reserved keywords

Python has some reserved keywords which have a special meaning, and cannot be used as variable names.

Python has some reserved keywords which have a special meaning, and cannot be used as variable names. Here they are:

```
False      class      finally    is         return
None       continue   for        lambda     try
True       def        from       nonlocal   while
and        del        global     not        with
as         elif       if         or         yield
assert     else       import     pass
break      except     in         raise
```

To summarise, the permitted variable names may contain letters, digits, underscore. Must begin with a letter. Cannot be the same as one of the reserved keywords.

To summarise, the permitted variable names may contain letters, digits, underscore. Must begin with a letter. Cannot be the same as one of the reserved keywords.

Example

Is there anything wrong with the following variables names?

To summarise, the permitted variable names may contain letters, digits, underscore. Must begin with a letter. Cannot be the same as one of the reserved keywords.

Example

Is there anything wrong with the following variables names?

```
lambda = 500
```

To summarise, the permitted variable names may contain letters, digits, underscore. Must begin with a letter. Cannot be the same as one of the reserved keywords.

Example

Is there anything wrong with the following variables names?

```
lambda = 500
```

This is a reserved keyword!

To summarise, the permitted variable names may contain letters, digits, underscore. Must begin with a letter. Cannot be the same as one of the reserved keywords.

Example

Is there anything wrong with the following variables names?

```
lambda = 500
```

This is a reserved keyword!

```
lecturer£name = "Kirill Sidorov"
```

To summarise, the permitted variable names may contain letters, digits, underscore. Must begin with a letter. Cannot be the same as one of the reserved keywords.

Example

Is there anything wrong with the following variables names?

```
lambda = 500
```

This is a reserved keyword!

```
lecturer£name = "Kirill Sidorov"
```

Inappropriate character!

To summarise, the permitted variable names may contain letters, digits, underscore. Must begin with a letter. Cannot be the same as one of the reserved keywords.

Example

Is there anything wrong with the following variables names?

```
lambda = 500
```

This is a reserved keyword!

```
lecturer£name = "Kirill Sidorov"
```

Inappropriate character!

```
10Forward = 42
```

To summarise, the permitted variable names may contain letters, digits, underscore. Must begin with a letter. Cannot be the same as one of the reserved keywords.

Example

Is there anything wrong with the following variables names?

```
lambda = 500
```

This is a reserved keyword!

```
lecturer£name = "Kirill Sidorov"
```

Inappropriate character!

```
10Forward = 42
```

Variable name cannot begin with a digit!

- Strings, values that represent bits of text, are just sequences of characters.
- There are lots of useful commands in Python to return information about and to manipulate strings.

- Strings, values that represent bits of text, are just sequences of characters.
- There are lots of useful commands in Python to return information about and to manipulate strings.

Example

`len("Kirill")` $\rightarrow$ 6.

- Strings, values that represent bits of text, are just sequences of characters.
- There are lots of useful commands in Python to return information about and to manipulate strings.

Example

`len("Kirill")` → 6.

`"Kirill".count("l")` → 2.

- We can use double quotes to denote a string:
  ```
  "Hello"
  ```
- Or single quotes:
  ```
  'Hello'
  ```
- Or triple double quotes:
  ```
  """Here
  is some
  text
  """
  ```

Example

What is wrong here? How can we fix this?

```
"She said "Hi" and smiled"
```

String concatenation

- Concatenation — joining two strings together, end to end.
- Use the `+` operator to concatenate strings.

String concatenation

- Concatenation — joining two strings together, end to end.
- Use the `+` operator to concatenate strings.

Example

```
"Abra" + "cadabra"  →  "Abracadabra"
```

Accessing individual characters

- Remember, strings are just sequences of characters.
- The subscription operator `[ ]` is used to access individual characters.

- Remember, strings are just sequences of characters.
- The subscription operator `[]` is used to access individual characters.

Example

Consider a variable called `my_string` which contains the string `"Spock"`. To find the third character of this string we would use:

```
first_officer = "Spock"
first_officer[3]    →    "c"
```

- Remember, strings are just sequences of characters.
- The subscription operator `[]` is used to access individual characters.

Example

Consider a variable called `my_string` which contains the string `"Spock"`. To find the third character of this string we would use:

```
first_officer = "Spock"
first_officer[3]    →    "c"
```

- We count from `0`!

Other operations on strings

- `s.lower()`, `s.upper()` returns the lowercase or uppercase version of the string.
- `s.strip()` returns a string with whitespace removed from the start and end.
- `s.isalpha()`, `s.isdigit()`, `s.isspace()` tests if all the string chars are in the various character classes.
- `s.startswith('other')`, `s.endswith('other')` tests if the string starts or ends with the given other string.
- `s.find('other')` searches for the given other string (not a regular expression) within s, and returns the first index where it begins or -1 if not found.
- `s.replace('old', 'new')` returns a string where all occurrences of `'old'` have been replaced by `'new'`.

See reference for more: `https://docs.python.org/3/library/stdtypes.html#string-methods`