

COMP1730/COMP6730

Programming for Scientists

Data: Values, types and expressions.



Lecture outline

- * Data and data types.
- * Expressions: computing values.
- * Variables: remembering values.

Example: Data analysis

- * *In 2018, enrolment in COMP1730/6730, at its peak, was 530 students. In 2019, the enrolment (so far) is 483 students. How big a decrease, in percent, is this?*
- * The decrease is: $530 - 483$
- * as a fraction of the 2018 number: $(530 - 483) / 530$
- * in percent: $((530 - 483) / 530) * 100$

Expressions

- * $((530 - 483) / 530) * 100$ is an *expression*;
- * it *evaluates* to `8.867924528301886`;
- * `483`, `530`, `100` and `8.867924528301886` are all *values*.
- * In interactive mode, the python interpreter will print the result of evaluating an expression:

```
>>> ((530 - 483) / 530) * 100
8.867924528301886
```

(with one exception, which we'll see later).

python syntax (recap)

- * A python program is a sequence of statements:
 - `import` a module;
 - function definition;
 - ~~function call~~ expression.
 - Every function call is an expression.
 - ...and more we'll see later.
- * Comment: `#` to end-of-line.
- * Whitespace:
 - end-of-line ends statement (except for function definition, which ends at the end of the suite);
 - indentation defines extent of (function) suite.

python expressions

- * Expressions are built up of:
 - constants (“literals”);
 - variables;
 - operators; and
 - function calls.
- * When an expression is executed, it *evaluates to a value* (a.k.a. the *return value*).
- * Expressions can act as statements (the return value is ignored), but statements cannot act as expressions.

Continuation

- * end-of-line marks the end of a statement.
- * *Except* that,
 - adding a “\” at the end makes the statement continue onto the next line, e.g.,

```
(2 ** 0) + (2 ** 1) + (2 ** 2) \  
+ (2 ** 3) + (2 ** 4)
```

- an expression enclosed in parentheses continues to the closing parenthesis, e.g.,

```
math.sqrt((x2 - x1) ** 2 +  
          (y2 - y1) ** 2)
```




Values and Types

Every value has a type

- * Value (data) types in python:
 - Integers (type `int`)
 - Floating-point numbers (type `float`)
 - Strings (type `str`)
 - Truth values (type `bool`)
 - ...and many more we'll see later.
- * Types determine what we can do with values (and sometimes what the result is).

- * The `type` function tells us the type of a value:

```
>>> type(2)
<class 'int'>
>>> type(2 / 3)
<class 'float'>
>>> type("zero")
<class 'str'>
>>> type("1")
<class 'str'>
>>> type(1 < 0)
<class 'bool'>
```

Numeric types

- * Integers (type `int`) represent positive and negative whole numbers (0, 1, 2, -1, -17, 4096, ...).
- * Values of type `int` have no inherent size limit.

```
>>> 2 ** (2 ** 2)
```

```
16
```

```
>>> 2 ** (2 ** (2 ** 2))
```

```
65536
```

```
>>> 2 ** (2 ** (2 ** (2 ** 2)))
```

```
...
```

- * Note: Can't use commas to “format” integers (must write `1282736`, not `1,282,736`).



- * Floating-point numbers (type `float`) represent decimal numbers.
- * Values of type `float` have limited range and limited precision.
 - Min/max value: $\pm 1.79 \cdot 10^{308}$.
 - Smallest non-zero value: $2.22 \cdot 10^{-308}$.
 - Smallest value > 1 : $1 + 2.22 \cdot 10^{-16}$.(These are typical limits; actual limits depend on the python implementation.)
- * Type `float` also has special values `± inf` (infinity) and `nan` (not a number).
- * More about floating-point numbers and their limitations in a coming lecture.

- * Every decimal number is a float:

```
>>> type(1.5 - 0.5)
```

```
<class 'float'>
```

```
>>> type(1.0)
```

```
<class 'float'>
```

- * The result of division is always a float:

```
>>> type(4 / 2)
```

```
<class 'float'>
```

- * floats can be written (and are sometimes printed) in “scientific notation”:

- $2.99e8$ means $2.99 \cdot 10^8$.

- $6.626e-34$ means $6.626 \cdot 10^{-34}$

- $1e308$ means $1 \cdot 10^{308}$.

Strings

- * Strings (type `str`) represent text.
- * A string literal is enclosed in single or double quote marks:

```
>>> "Hello world"
```

```
'Hello world'
```

```
>>> '4" long'
```

```
'4" long'
```

- A string can contain other types of quote mark, but not the one used to delimit it.
- * More about strings in a coming lecture.

Type conversion

- * Explicit conversions use the type name like a function:

```
>>> int(2.0)
>>> float("-1.05")
>>> str(0.75 * 1.75)
```

- * Conversion from `str` to number only works if the string contains (only) a numeric literal.
- * Conversion from `int` to `float` is automatic.
 - E.g., `int times float` becomes a `float`.



Expressions: Operators and Functions

Numeric operators in python

<code>+, -, *, /</code>	standard arithmetic
<code>**</code>	power (<code>x ** n</code> means x^n)
<code>//</code>	floor division
<code>%</code>	remainder

- * Some operators can be applied also to values of other (non-numeric) types, but with a different meaning (this is called “operator overloading”).
- * We’ll see more operators later in the course.

Precedence

- * There is an order of precedence on operators, that determines how an expression is read:
 - $2 * 3 - 1$ means $(2 * 3) - 1$, not $2 * (3 - 1)$.
 - $-1 ** 5$ means $-(1 ** 5)$, not $(-1) ** 5$.
- * Operators with equal precedence associate left:
 - $d / 2 * pi$ means $(d / 2) * pi$, not $d / (2 * pi)$
- * ...except exponentiation, which associates right.
- * Whenever it is not obvious, *use parentheses to make it clear.*

Math functions

- * The `math` module provides standard math functions, such as square root, logarithm, trigonometric functions, etc.

```
>>> import math
>>> help(math) # read documentation
...
>>> math.sqrt(3 ** 2 + 4 ** 2)
5.0
```

- * Almost all math functions take and return values of type `float`.

Comparison operators

<, >, <=, >=	ordering (strict and non-strict)
==	equality (note double '=' sign)
!=	not equal

- * Can compare two values of the same type (for almost any type).
- * Comparisons return a *truth value* (type `bool`), which is either `True` or `False`.
- * *Caution:* Conversion from any type to type `bool` happens automatically, but the result may not be what you expect.



Variables

Variables

- * A *variable* is a name that is associated with a value in the program.
 - The python interpreter stores name–value associations in a *namespace*.
(More about namespaces later in the course.)
- * A variable can be an expression: evaluating it returns the associated value.
- * A name–value association is created by the first *assignment* to the name.

Valid names in python (reminder)

- * A (function or variable) name in python may contain letters, numbers and underscores (_), but must begin with a letter or underscore.
- * Reserved words cannot be used as names.
- * Names are *case sensitive*: upper and lower case letters are not the same.
 - `Length_Of_Rope` and `length_of_rope` are different names.

Variable assignment

- ★ A variable assignment is written

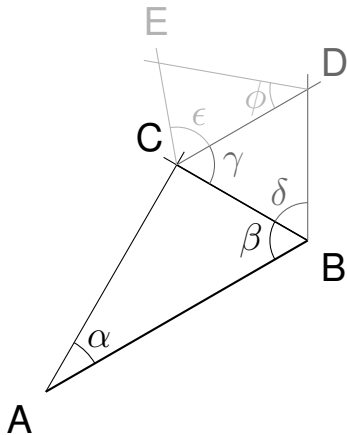
var_name = expression

- Reminder: Equality is written `==` (two `=`'s).
- Assignment is a statement.
- ★ When executing an assignment, the interpreter
 1. evaluates the right-hand side expression;
 2. associates the left-hand side name with the resulting value.

Programming problem

- ★ *Triangulation:*
Given the distance AB (the *baseline*) and the angles to a common landmark C , calculate a new baseline

$$BC = AB \frac{\sin \alpha}{\sin(\alpha + \beta)}$$



The `print` function

- * `print` prints text to the console:

```
>>> print("The answer is:", 42)
The answer is: 42
```

- Non-text arguments are converted to type `str` before printing.
- `print` takes a variable number of arguments, and prints them all followed by a newline.
- * Print the result, and intermediate steps, when a program is run in script mode.