Python

A quickstart into the key concepts of programming

Built-in atomic data types

Key concepts in programming

- Variables (*integers, strings, dates, etc.)*
- Flow control (*if then, loop, etc.*)
- Functions (*list of steps the code will follow*)

Built-in atomic data types

basic_datatypes.ipynb

Built-in types

Tyoe	Example	Description

int	`In [1]: x = 1`
	`type(x)`
	`Out [1]: int`
		Integers (i.e., whole numbers)

float	`x = 1.0`	Floating-point numbers (i.e., real numbers)
complex	`x = 1 + 2j`	Complex numbers (i.e., numbers with a real and imaginary part)
bool	`x = True`	Boolean: True/False values
str	`x = 'abc'`	String: characters or text
NoneType	`x = None`	Special object indicating nulls

Integer

- Most basic numerical type.
- Any number without a decimal point is an integer.
- Note: Python integers are variable-precision, not limited as in C, Matlab to 4 or 8 bytes.
- `2**200 # is possible`

Float point number

- The floating-point type can store fractional numbers.
- standard decimal notation, or in exponential notation

```
x = 0.000005
y = 5e-6
```

- Note: limited precision

```
0.1 + 0.2 == 0.3
False
```

- *Tip: never* rely on exact equality tests with floating-point values.

Complex Numbers: j

- A complex number consists of 2 doubles:

```
complex(1, 2)
c1 = 3 + 5.3j
c1.imag
5.3
c1.real
3.0
c2 = 3.3 + a*1j
```

- It accepts either J or j but the numerical value of the imaginary part must immediately precede it. If the imaginary part is a variable as in these examples, the 1 must be present.

Boolean

- Simple type with two possible values: True and False (capital T and F!)

```
result = (4 < 5)
result
True
type(result)
bool
```

- Booleans can be constructed using the `bool()` object constructor

```
x = ' '
y = 15
print(bool(x))
print(bool(y))
True
True
```

Type conversions

- If a variable is of one type but it needs to be of a different type, it is necessary to do a *type conversion* aka a *cast*.

```
R=float(I)
I=int(R)
Z=complex(r1,r2)
```

- Converting an integer to a string: `str`

Strings

- A *string* is a (ordered) sequence of characters
- Created with single ‘ or double quotes “
- Many useful string functions and methods
  - Check with `dir`
- Strings are *immutable* and cannot be changed. They can only be overwritten.

```
a = 'help'
a[1] = ‘a’
```

```
TypeError  Traceback (most recent call last)
<ipython-input-43-294a43332c98> in <module>
      1 a = 'help'
----> 2 a[1] = 'a'
```

- Operators: + and `[:]` (Concatenation and Slicing)

Strings

- Some useful methods
- Syntax: `<string name>.<method name>(…)`
- `S = 'Hello String'`
- `S.upper()`: transform to upper case
- `S.index(sub)`: position of the first occurence of sub in S
- `S.count(sub)`: number of times sub appears inside S
- `S.strip()`: Returns a copy of S with white-space removed at ends
- *File: string_intro.py*

Docstrings

- Documentation strings or "docstrings" use a special form of comment.
- The lines are enclosed in triple double quotes

""" """

- Everything within triple double quotes is treated as a literal string and a comment, including line breaks.
- Docstrings are placed at the top of program units, just under the declaration of the unit name (if present).
- If they are correctly placed, certain automated tools are available to display the documentation.

None

- A special type, the NoneType, which has only a single possible value: None.

```
In [24]: type(None)
Out [24]: NoneType
```

- Most commonly used as the default return value of a function.

```
In [25]: return_value = print('abc')
abc
In [26]: print(return_value)
None
```