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

Type	Example	Description

int	`x = 1`
	`type(x)`
	`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

```
print(bool(''))
False
print(bool(' '))
True
```

Boolean

- The numerical values of True and False
- They have numerical values:
  - True: 1
  - False: 0

```
True == 1
True
False == 0
True
```

Strings

- A *string* is a (ordered) sequence of characters.
  - Behind the scenes strings are stored as a tuple of letters
- Created with single ‘ or double quotes “
- Strings enclosed in triple quotes (""" or ''') can also be block strings: they will encode newline characters if the string is entered over multiple lines. In addition, they are conventionally used to create docstrings (documentation strings) within source code.
- Many useful string functions and methods
  - Check with `dir`

Strings

- 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(+), multiplication 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*

Format strings (f-strings)

- Available since Python 3.6
- F-string is a string literal that is prefixed with `f` or `F`. These strings may contain replacement fields (delimited by curly braces {} – *fill out the braces*). F-string is evaluated at run time.

```
name = 'Peter'
age = 23
print('%s is %d years old' % (name, age))
print('{} is {} years old'.format(name, age))
print(f'{name} is {age} years old’)
```

- *File: fstring_01.py*
- https://realpython.com/python-string-formatting/

Format strings (f-strings)

https://www.pythoncheatsheet.org/cheatsheet/string-formatting

Number	Format	Output	description
3.1415926	{:.2f}	3.14	Format float 2 decimal places
3.1415926	{:+.2f}	+3.14	Format float 2 decimal places with sign
-1	{:+.2f}	-1.00	Format float 2 decimal places with sign
2.71828	{:.0f}	3	Format float with no decimal places
4	{:0>2d}	04	Pad number with zeros (left padding, width 2)
4	{:x<4d}	4xxx	Pad number with x’s (right padding, width 4)
10	{:x<4d}	10xx	Pad number with x’s (right padding, width 4)
1000000	{:,}	1,000,000	Number format with comma separator
0.35	{:.2%}	35.00%	Format percentage
1000000000	{:.2e}	1.00e+09	Exponent notation
11	{:11d}	11	Right-aligned (default, width 10)
11	{:<11d}	11	Left-aligned (width 10)
11	{:^11d}	11	Center aligned (width 10)

Character

- `ord()` takes a string argument of a single Unicode character and returns its integer Unicode code point value.

```
ord('a’)
97
```

- `chr()` function takes integer argument and returns the string representing a character.

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.

```
type(None)
NoneType
```

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

```
return_value = print('abc')
abc
print(return_value)
None
```

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)
```

Type conversions

Function	Converting to		Function	Converting to
`int(y)`	an integer.		`tuple(y)`	a tuple.
`float(y)`	a floating-point number.		`list(y)`	a list.
`str(y)`	a string.		`set(y)`	a set.
`ord(y)`	a character into an integer.		`dict(y)`	creates a dictionary and *y* should be a sequence of (key, value) tuples.
`chr(y)`	an integer into a character.		`complex(real [imag])`	creates a complex number.
`hex(y)`	an integer to a hexadecimal string.			
`oct(y)`	an integer to an octal string.			