Variables and Data Types

Learning objects

- **Types of Numbers**: Identify integers, floats, and complex numbers in Python.

- **Expressions**: Write and evaluate basic mathematical expressions.

- **Booleans**: Use Boolean values (`True`, `False`) in logical operations.

- **Conversions**: Convert between data types (e.g., strings to integers).

- **Mutable vs Immutable**: Differentiate between mutable and immutable objects.

- **Functions vs Methods**: Understand the difference between standalone functions and object-specific methods.

- **User Input and `eval()`:** Collect user input and evaluate it with `eval()`.

Review

Here's a short DNA sequence:
ACTGATCGATTACGTATAGTATTTGCTATCATACATATAT
ATCGATGCGTTCAT

Write a program that will count the number of As, Ts and the total length of the sequence

Several Types of Numbers

- Numbers have two main types

  - Integers are whole numbers:
    -14, -2, 0, 1, 100, 401233

  - Floating Point Numbers have decimal parts: -2.5 , 0.0, 98.6, 14.0

- There are other number types - they are variations on float and integer

```
>>> xx = 1
>>> type(xx)
< ass 'int'>
>>> temp = 98.6
>>> type(temp)
< ass'float'>
>>> type(1)
< ass 'int'>
>>> type(1.0)
< ass'float'>
>>>
```

Expressions

Numeric Expressions

- We use "computer-speak" to express the classic math operations
- Asterisk is multiplication
- Exponents (raise to a power)

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

Numeric Expressions

```
>>> xx = 2
>>> xx = xx + 2
>>> print(xx)
4
>>> yy = 440 * 12
>>> print(yy)
5280
>>> zz = yy / 1000
>>> print(zz)
5.28
```

```
>>> jj = 23
>>> kk = jj % 5
>>> print(kk)
3
>>> print(4 ** 3)
64
```

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

Operator Precedence Rules

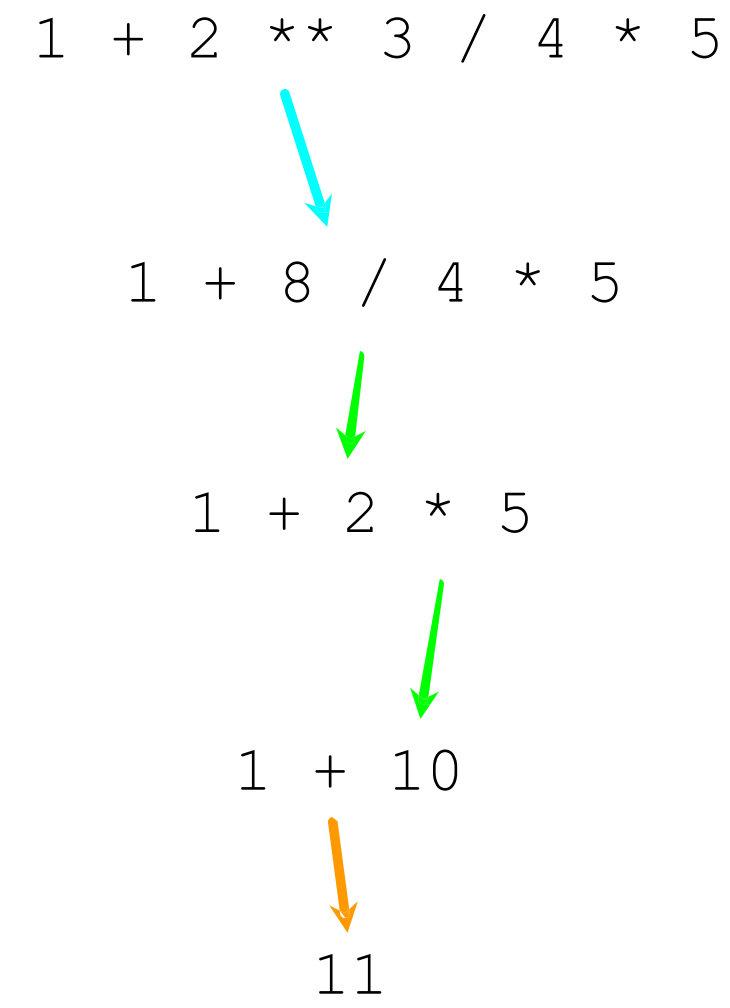
Highest precedence rule to lowest precedence rule:

- Parentheses are always respected
- *function() call, execute a function and get what it returns*
- Exponentiation (raise to a power)
- Multiplication, Division, and Remainder
- Addition and Subtraction
- Left to right
- *Assignment =*

Parenthesis
Power
Multiplication
Addition
Left to Right

```
>>> x = 1 + 2 ** 3 / 4 * 5
>>> print(x)
11.0
>>>
```

Parenthesis
Power
Multiplication
Addition
Left to Right



Reserved Words

You cannot use reserved words as variable names / identifiers

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

```
with = 'test'
```

```
$ python3 test.py
  File "test.py", line 1
    with = 'test'
         ^
SyntaxError: invalid syntax
```

There are 35 keywords in Python 3.8 - This number can vary slightly in the course of time
All the keywords except `True`, `False` and `None` are in lowercase
Python 3.9 added **`__peg_parser`**

Booleans

- Type that represents boolean values
- It is named after George Boole, English mathematician and logician
- `Basis of computer circuits`
- The data type has exactly two values: **`True`** and **`False`**
- No quotes! They are not strings
- Case sensitive as usual: capital **`True`** and **`False`**
- You can’t do arithmetic with the values
- The operators used with them are: **`and`**, **`or`** and **`not`**
- Mostly used in **`if`** and **`while`** statements

```
>>> type(True)
<class 'bool'>
>>> 5 == (3 + 2)
True
>>> 5 == 6
False
>>> j = "hel"
>>> j + "lo" == "hello"
True
```

Different ways to access functionality

- Operators

  Special symbols to denote an operation (eg + * / etc)

  5 + 10

- Functions

  Named pieces of functionality into which data is passed

  `len("simon")`

- Methods

  –Functions which are accessed via the data directly

  `"simon".upper()`

Functions vs Methods

- Functions
  - –Named pieces of code. All data (arguments) must be passed in to them. Accessed either in the core language or from packages

```
>>> len("Simon")
5
```

- Methods
  - –Functions which are associated with a type of data (string, date etc). Called via the data, you don’t need to pass the data in to the method

```
>>> "Simon".upper()
'SIMON'
```

Functionality is linked to data type

```
5 + 10                  #15
“5” + “10”              #510

“5” .upper()            #5
5.upper()               # SyntaxError: invalid syntax


float (“5”) + int(10)       #15
str(5) + str(10)            #510
```

Mutable

Immutable vs Mutable

- An **immutable** variable cannot be changed after it is created (with one caveat)
- If you wish to change an **immutable** variable, such as a string, you must create a new instance and bind the variable to the new instance
- A **mutable** variable can be changed in place
- Refer to this list of the Python data types, and whether they are mutable

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

- We'll come back to this Lecture 2 for a deeper dive

but... Tuples are “immutable”

Unlike a list, once you create a tuple, you cannot alter its contents - similar to a string

```
>>> x = [9, 8, 7]
>>> x[2] = 6
>>> print(x)
>>>[9, 8, 6]
>>>
```

List

```
>>> protein = "vlspadktnv“
>>> protein[2] = 'D'
Traceback:'str'
object does
not support item
Assignment
>>>
```

String

```
>>> z = (5, 4, 3)
>>> z[2] = 0
Traceback:'tuple'
object does
not support item
Assignment
>>>
```

Tuple

User `input` and `eval`

User `input`

- We can instruct Python to pause and read data from the user using the `input()` function

  Pauses the execution of the program, displaying a blinking cursor. Waits for the user to press Enter

- The `input()` function returns a string, the entire line of input that the user typed, **without the newline at the end, as a string**

- If the user just pressed `Enter` without typing anything, it returns an empty string

```
nam = input('Who are you? ')
print('Welcome', nam)
```

```
Who are you? Chuck
Welcome Chuck
```

Converting User `input`

- If we want to read a **`number`** from the user, we must convert it from a string to a number using a type conversion function
- Use **Type conversion**
- What if the input cannot be converted properly to a number?
- Run-time error (`ValueError` exception)

```
inp = input('Europe floor?')
usf = int(inp) + 1
print('US floor', usf)
```

```
Europe floor? 0
US floor 1
```

User `input`

```
# read_name.py

name = input("Enter your name: ")
print("Welcome", name)  # note the space

age = input("Enter your age: ")
print(type(age))
age = int(age) + 1

print("Hello ", name)
print("On your next birthday, you will be ", age)
```

```
Enter your name: John
Welcome John
Enter your age: 30
<class 'str'>
Hello  John
On your next birthday, you will be  31
```

- We'll deal with bad input data later on…
- But feel free to read ahead: Errors and Exceptions
  https://docs.python.org/3/tutorial/errors.html

`eval()`

- The `eval()` method parses the expression passed to this method and runs python expression (code) within the program
- The `eval()` method returns the result evaluated from the expression

```
x = 1

print(eval('x + 1'))
```

2