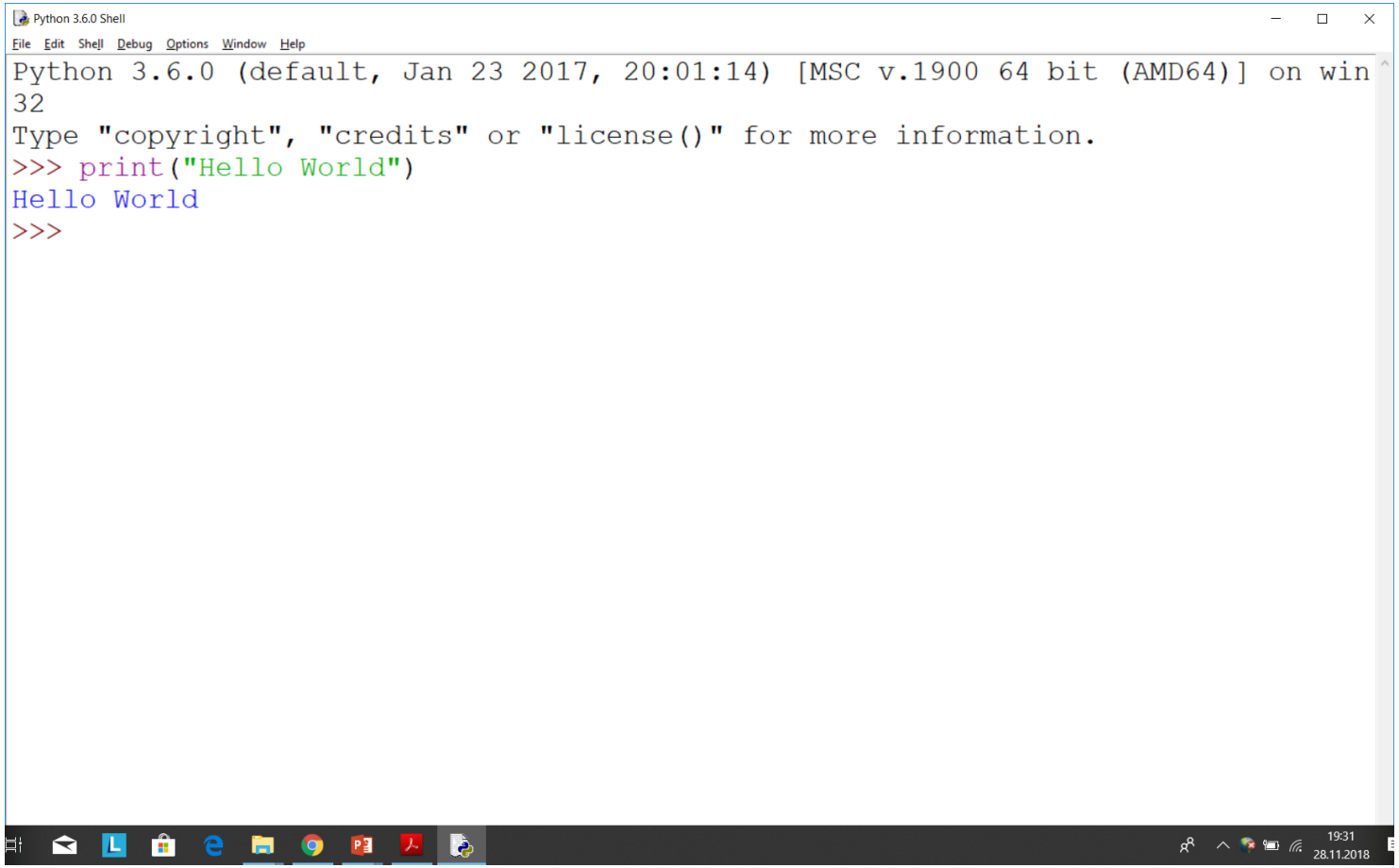
Python

Introduction, Data Types

Running Python

- Python is an interpreted language
  - A language in which the instructions are executed by an “interpreter” at run time
- As an interpreted language, Python is relatively easy to port to many environments to be used by various interpreters
  - Command line
    - Via the Python interpreter
  - Script files
    - Text files containing Python commands

Interpreter command line



Script file

Python 3.6.0 Shell

File Edit Shell Debug Options Window Help

```
Python 3.6.0 (default, Jan 23 2017, 20:01:14) [MSC v.1900 64 bit (AMD64)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>>
=========== RESTART: C:\Users\özlem\Desktop\python\hafta1\hello.py ===========
Hello World!
>>>
```

hello.py - C:\Users\özlem\Desktop\python\hafta1\hello.py (3.6.0)

File Edit Format Run Options Window Help

```
print("Hello World!")
```

Python

- The Python language itself is fairly simple, comprised of only 33 keywords

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	

- The power comes from the large number of built-in functions and the wealth of available libraries

Numeric Values

- **Recall**: the purpose of a computer is to compute some value based on some input
- Python supports several numeric data types
  - Integers in four bases
    - Base 10 (decimal): 432
    - Base 2 (binary): 0b110110000
    - Base 8 (octal): 0o660
    - Base 16 (hexadecimal): 0x1B0
    - Range is limited only by hardware
  - Floating point numbers
    - Can be expressed as decimal floating point: 432.5
    - Or Scientific Notation: 4.325e2
      - e stands for exponent, 4.325e2 is the same as 4.325×10^2
    - Range is about (-1.8e308 – 1.8e308)
      - Higher than 1.79e308 is inf.
      - Lower than -1.79e308 is -inf
  - Complex numbers
    - (real)+(imaginary)j, i.e. 4+32.5j

String Values

- Strings are sequences of characters
- Strings in Python can be enclosed in either single quotes (') or double quotes ("), or three of each (''' or """)
- '''Hello World''', "Hello World" and 'Hello World' are equivalent
  - Number in quotes are treated as strings, not numbers
  - ‘42.0’ does not equal 42.0

String Values

- Double quoted strings can contain single quotes inside them, as in "Bruce's beard", and single quoted strings
- can have double quotes inside them, as in 'The knights who say "Ni!"'.
- Strings enclosed with three occurrences of either quote symbol are called triple quoted strings. They can contain either single or double quotes:

```
>>> print('''"Oh no", she exclaimed, "Ben's bike is broken!"''')
"Oh no", she exclaimed, "Ben's bike is broken!"
>>> |
```

String Values

- Triple quoted strings can even span multiple lines:

```
>>> message = """This message will
... span several
... lines."""
>>> print(message)
This message will
... span several
... lines.
>>> |
```

Boolean Values

True or False

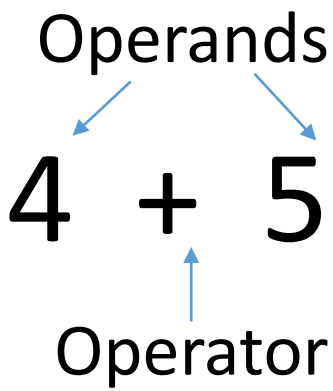
Operators

- Python can act as a simple calculator and support many operators

Sr.No	Operator & Description
1	** Exponentiation (raise to the power)
2	~ + - Complement, unary plus and minus
3	* / % // Multiply, divide, modulo and floor division
4	+ - Addition and subtraction
5	>> << Right and left bitwise shift
6	& Bitwise 'AND'
7	^ | Bitwise exclusive `OR' and regular `OR'
8	<= < > >= Comparison operators
9	<> == != Equality operators
10	= %= /= //= -= += *= **= Assignment operators
11	is is not Identity operators
12	in not in Membership operators
13	not or and Logical operators

Using Operators

- Based on arithmetic, though with more operators. Follows the precedence table
- Parenthesis overrides precedence
  - What's the difference between 4 + 5 * 3 and (4 + 5) * 3?
- Note! Exponentiation ** has higher precedence than unary –
  - What does -1**2 equal?



String Operators

+ Concatenation

- “Hot” + “dog” is “Hotdog”

* Replication

- “Clap” * 3 is “ClapClapClap”

Variables

- A variable is a name which refers to a value. That value can vary over time.
- a = 5
  - Create an integer with value 5 and then associates the name **a** to it
- pi = 3.14159
  - Create a float with value 3.14159 and then associate the name pi to it
- Before a variable can be used, it must be assigned a value

```
>>> x = 5
>>> y = 6
>>> z = x + y
>>> print(z)
11
>>> y = y + 6
>>> print(y)
12
>>> y = w + 7
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    y = w + 7
NameError: name 'w' is not defined
>>> |
```

Variable names

- Letters, numbers, and underscore _
- Python is case sensitive
  - a and A are different values
- Cannot start with a number
  - season5 is valid, 5season is not
- Cannot be one of the Python reserved words
  - lambda is not a valid variable name
- Make variable names meaningful
  - Storing the area of a triangle
    - x is terrible, what does it have to do with area or a triangle
    - a is ok, but not descriptive
    - area is better, area_of_a_triangle is valid but probably too long
    - t_area or triArea may be better depending on the context

```
>>> 76trombones = "big parade"
SyntaxError: invalid syntax
>>> more$ = 1000000
SyntaxError: invalid syntax
>>> class = "Computer Science 101"
SyntaxError: invalid syntax
```

Assignments

$$x = 4 * 7 + a * 2$$

Left Hand Value (LHV)

The name the value will be stored under

Right Hand Value (RHV)

The value being generated

Assignment Operator

Defines how will the value be stored

Assignments

$$x = 4 * 7 + a * 2$$

Left Hand Value (LHV)

The name the value will be stored under

Right Hand Value (RHV)

The value being generated

Assignment Operator

Defines how will the value be stored

Operator	Action
=	Simple Assignment
+=	Add RHV to LHV
-=	Subtract RHV from LHV
*=	Multiply RHV to LHV
/=	Divide LHV by RHV
…	Etc.

Assignments

$$x = 4 * 7 + a * 2$$

Left Hand Value (LHV)

The name the value will be stored under

Right Hand Value (RHV)

The value being generated

Assignment Operator

Defines how will the value be stored

```
x = 10
x += 35.5
print(x)
```

```
45.5
```

Operator	Action
=	Simple Assignment
+=	Add RHV to LHV
-=	Subtract RHV from LHV
*=	Multiply RHV to LHV
/=	Divide LHV by RHV
…	Etc.

Multiple Assignment

- Assigning more than one variable at a time is possible
  - a, b, c = ‘foo’, ‘bar’, ‘baz’
- Swapping variable values
  - x, y = y, x

Variable Type

- Unlike C, C++, or other languages, Python is dynamically typed
- If a variable comes into existence by assigning it a value, how does Python know what data type it is?

Variable Type

- Unlike C, C++, or other languages, Python is dynamically typed
- If a variable comes into existence by assigning it a value, how does Python know what data type it is?
    - It infers the type from the assignment

```
In [15]: x = 5
         print(type(x))

         <class 'int'>

In [16]: x = 5.5
         print(type(x))

         <class 'float'>
```

Variable Type

- Unlike C, C++, or other languages, Python is dynamically typed
- If a variable comes into existence by assigning it a value, how does Python know what data type it is?
  - It infers the type from the assignment
  - integers are preferred over floats

```
In [15]: x = 5
         print(type(x))

         <class 'int'>

In [16]: x = 5.5
         print(type(x))

         <class 'float'>

In [17]: x = 5
         print(type(x))
         x = 5.5
         print(type(x))

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

Variable Type

- Unlike C, C++, or other languages, Python is dynamically typed
- If a variable comes into existence by assigning it a value, how does Python know what data type it is?
  - It infers the type from the assignment
  - integers are preferred over floats
  - the type of a variable dynamically changes based on need

```
In [15]: x = 5
         print(type(x))

         <class 'int'>

In [16]: x = 5.5
         print(type(x))

         <class 'float'>

In [17]: x = 5
         print(type(x))
         x = 5.5
         print(type(x))

         <class 'int'>
         <class 'float'>

In [20]: x = 1/2
         print(x)
         print(type(1))
         print(type(2))
         print(type(x))

         0.5
         <class 'int'>
         <class 'int'>
         <class 'float'>
```

Converting to Strings

- Use the repr(value) or str(value) function to convert a number to a string
  - repr – string **repr**esentation

```
>>> x = 1/2
>>> y = ' of the time'
>>> print(x + y)
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    print(x + y)
TypeError: unsupported operand type(s) for +:
'float' and 'str'
>>> print(repr(x) + y)
0.5 of the time
>>> |
```

Comments

- Text put into a program to assist the programmer
- All text which follows a # is ignored by Python

```
x = 5
y = 7
if x < y:
    print(x) # print the lower of x and y
else:
    print(y)
```

Bugs

- Bug
  - An error in a program
- Types of bugs
  - Syntax errors
    - Problems with the syntax of the source code which stops the parser from being able to understand the program
  - Run-time errors
    - Errors which occur while the program is running.
  - Semantic (Logic) errors
    - Programs which parse and run fine, but do not do what is intended
- Debugging
  - The process of removing bugs

Syntax Error

- An error in the grammar of the program

```
>>> x = 5
>>> pirnt(x)
Traceback (most recent call last):
  File "<pyshell#39>", line 1, in <module>
    pirnt(x)
NameError: name 'pirnt' is not defined
>>> |
```

Run-time Error

- An error which occurs while the program is running which the stops the operation of the Python interpreter

```
x = 1/2
y = ' of the time'
print(x)
print(x + y)
```

```
0.5
Traceback (most recent call last):
  File "C:/Users/özlem/Desktop/python/hafta1/e
xample-run-time.py", line 4, in <module>
    print(x + y)
TypeError: unsupported operand type(s) for +:
'float' and 'str'
```

Semantic (Logic) Error

- An error in the logic of a program, causing the program to generate a result different than expected

```
In [30]: x = 5
         y = 7
         if x > y:      # print the lower of x and y
             print(x)
         else:
             print(y)

         7
```

- Note how Python helps find the source of Syntax and Run-time errors, but not Logic errors

Resources

- Downey, A. (2016) *Think Python, Second Edition* Sebastopol, CA: O'Reilly Media
- Wentworth P., Elkner J., Downey A., and Meyers C. How to Think Like a Computer Scientist: Learning with Python 3
- Bryan Burlingame's course notes