LESSON 1

Administrative . Computers . Hardware Basics .
Introduction to Python Programming

Create an account in ri.coursemology.org

- Go to your email (mostly RI email, some gmail and Hotmail)
- Accept the invitation
- Create an account using your email and chosen password
- Make sure that the URL is ri.coursemology.org not just coursemology.org

COMPUTER . HARDWARE BASICS

PYTHON

Assigning value to variable

Valid identifiers	Invalid identifiers	Explanation
`sum_of_scores`	`sum-of-scores`	Identifiers cannot contain the hyphen character "-"
`my_class`	`class`	`class` is a reserved word
`BOX_SIZE`	`size_correct?`	Identifiers cannot contain the question mark character "?"
`test_score_3`	`3rd_test_score`	The first character of an identifier cannot be any of the digits "0" to "9"
`oneword`	`two words`	Identifiers cannot contain any spaces

- Must start with alphabets or underscore (_)
- Can contain numbers 0 to 9
- Case sensitive, bruce and Bruce are two different variables
- Does **not** contain spaces, punctuations
- Does not use Python **keywords**, see right

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

Data Types

Data type	Python	Valid values	Example
Integer	`int`	Whole numbers	`1234`
Floating point	`float`	Real numbers	`12.34`
String	`str`	Text (can include digits)	`"ABC123"`
Boolean	`bool`	True, False	`True`

numbers – integer and float

- There is also a limit to the most positive and most negative number that can be represented using a float.
- In most versions of Python, the most positive number is 1.79×10^{308} and the most negative number is -1.79×10^{308}.
- Beyond that, inf or – inf
- 1e3 = 1×10^{3}

Operator	Name	Description	Examples
`+`	Addition	Returns the sum of two values	`>>> 2017 + 1e3` `3017.0`
`-`	Subtraction	Returns the difference of two values	`>>> 2017 - 1e3` `1017.0`
`*`	Multiplication	Returns the product of two values	`>>> 2017 * 1e3` `2017000.0`
`/`	Division	Returns the value on the left divided by the value on the right	`>>> 2017 / 1e3` `2.017`
`//`	Floor division	Returns the value on the left divided by the value on the right, rounded down to the nearest integer (Note that the data type of the result may not necessarily be an `int`.)	`>>> 2017 // 1e3` `2.0`
`**`	Exponentiation (power)	Returns the value on the left raised to the power of the value on the right	`>>> 2 ** 1e3` `1.07150860719e+301`
`%`	Modulus (remainder)	Returns the remainder when the value on the left is divided by the value on the right	`>>> 2017 % 1e3` `17.0`

Comparing

Operator	Name	Description	Examples

==	Equivalence	Returns the `bool` value `True` if the two values are equivalent and `False` if they are not	`>>> 2017 == 2017`
			`True`
			`>>> 2017 == 2018`
			`False`

!=	Non-equivalence	Returns the `bool` value `False` if the two values are equivalent and `True` if they are not	`>>> 2017 != 2017`
			`False`
			`>>> 2017 != 2018`
			`True`

Operator	Name	Description	Examples

<	Less than	Returns the `bool` value `True` if the value on the left is less than the value on the right and `False` if it is not	`>>> 2017 < 2017`
			`False`
			`>>> 2017 < 2018`
			`True`
			`>>> 2017 < 20.17`
			`False`

<=	Less than or equal to	Returns the `bool` value `True` if the value on the left is less than or equal to the value on the right and `False` if it is not	`>>> 2017 <= 2017`
			`True`
			`>>> 2017 <= 2018`
			`True`
			`>>> 2017 <= 20.17`
			`False`

>	Greater than	Returns the `bool` value `True` if the value on the left is greater than the value on the right and `False` if it is not	`>>> 2017 > 2017`
			`False`
			`>>> 2018 > 2017`
			`True`
			`>>> 2017 > 20.17`
			`True`

>=	Greater than or equal to	Returns the `bool` value `True` if the value on the left is greater than or equal to the value on the right and `False` if it is not	`>>> 2017 >= 2017`
			`True`
			`>>> 2018 >= 2017`
			`True`
			`>>> 2017 >= 20.17`
			`True`

Common String escape codes

Escape code	Meaning
`\\`	Backslash (\)
`\'`	Single-quote (')
`\"`	Double-quote (")
`\n`	Newline character
`\t`	Tab character
`\`	Ignore end of line

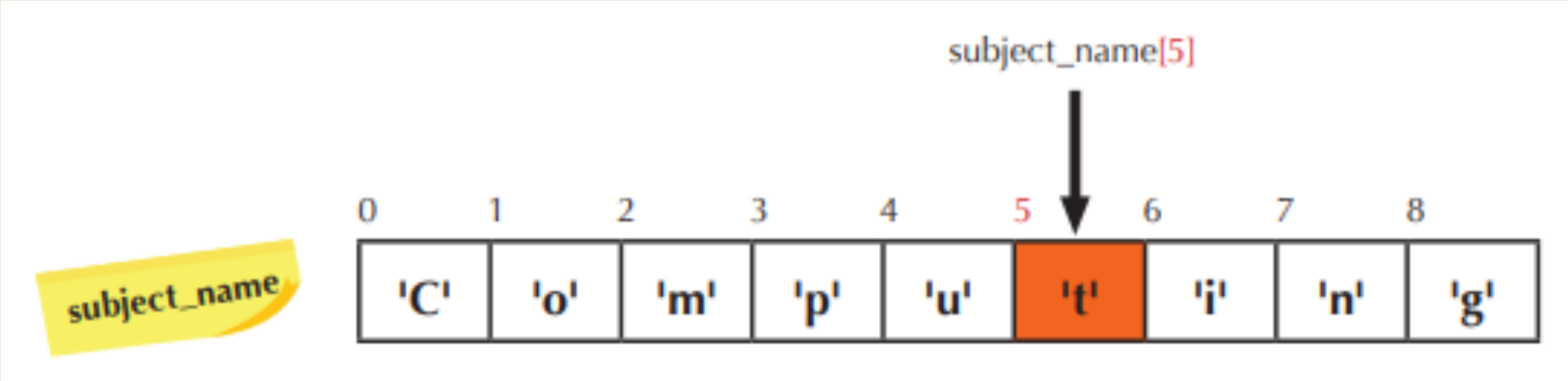
string operators

Operator	Name	Description	Examples
+	Concatenation	Joins the sequence on the right onto the end of the sequence on the left	`>>> print("Computing" + "2017")` `Computing2017`

*	Repetition	Repeats the contents of a sequence a number of times	`>>> print("Computing" * 3)`
			`ComputingComputingComputing`
			`>>> print(3 * "2017")`
			`201720172017`

string indexing

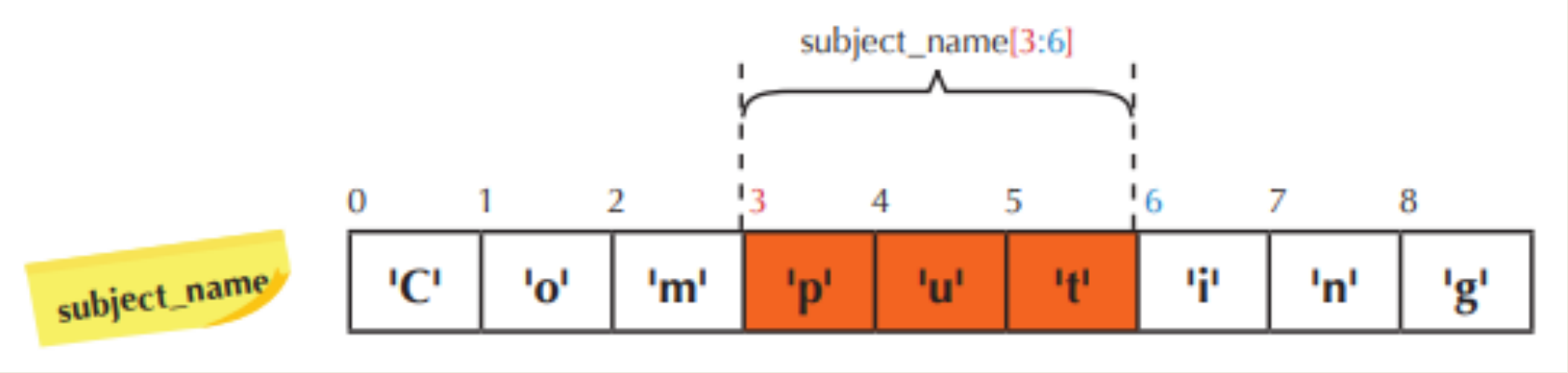
`sequence_name[i]` , where i is an integer value. This integer value is called the index.



string slicing

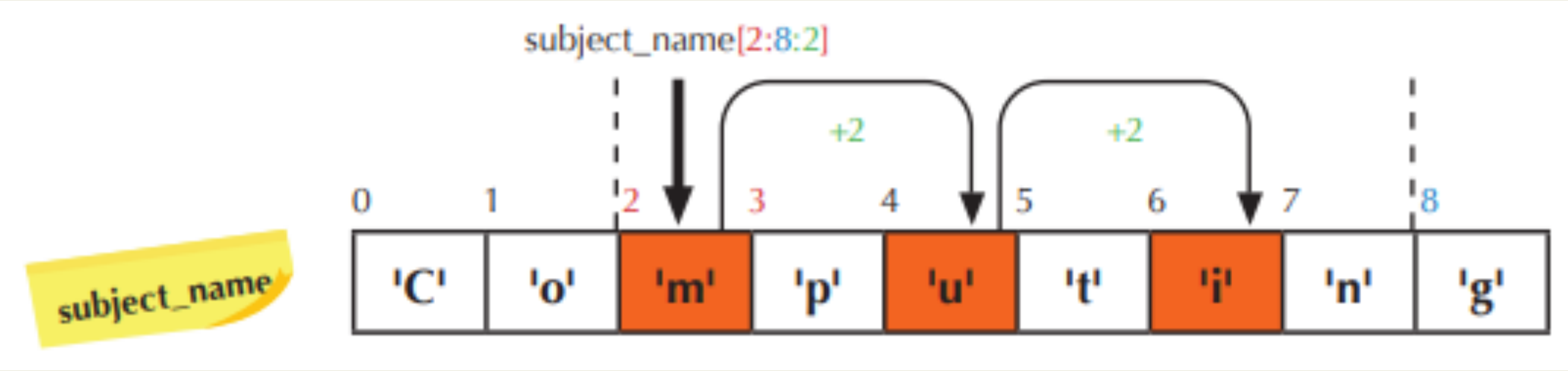
`sequence_name[a : b]` , where a and b are integer values indicating the start and stop indices respectively.

This extracts the sequence of values or characters positioned from the start index up to but **not including the stop index**.



slice with step

`sequence_name[a : b : c]` , where a and b are the start and stop indices while c is called the step.



Operator	Name	Description	Examples

`[i]`	Index	Returns the value in the `i`-th position of the sequence	`>>> print("Computing"[0])`
			`C`
			`>>> print("Computing"[3])`
			`p`
			`>>> print("Computing"[-1])`
			`g`

`[a:b]`	Slice	Returns a sequence of values starting from the value at index `a` up to but not including the value at index `b`; `a` is treated as 0 if omitted and `b` is treated as the length of the sequence if omitted	`>>> print("Computing"[3:6])`
			`put`
			`>>> print("Computing"[:7])`
			`Computi`
			`>>> print("Computing"[3:])`
			`puting`

`[a:b:c]`	Slice with step	Returns a sequence of values starting from the value at index `a` up to but not including the value at index `b`, in increments of `c`; `a` is treated as 0 if omitted, `b` is treated as the length of the sequence if omitted and `c` is treated as 1 if omitted.	`>>> print("Computing"[2:8:2])`
			`mui`
			`>>> print("Computing"[:8:2])`
			`Cmui`
			`>>> print("Computing"[2::2])`
			`muig`
			`>>> print("Computing"[2:8:])`
			`mputin`

function

Syntax 4.2 Function Call

```
function_name()
```

```
function_name(argument_1)
```

```
function_name(argument_1, argument_2)
```

built-in function

- Common in-built function:

```
print() , len() , type() , input() , help() , range() ,
int() , float() , str() , … etc
```

self-defined function

```
def function_name (parameter1 , parameter2 , …):
      instructions
      …
      (return values optional)
```

Eg.

```
def square(num):
      return num * num
```

```
def print_3(msg):
      print(msg)
      print(msg)
      print(msg)
```

Look up & Read up

- Why 0.3 + 0.3 + 0.3 ≠ 0.9 ?
- Binary number system
- Denary number system
- Hexadecimal number system
- Conversion between them